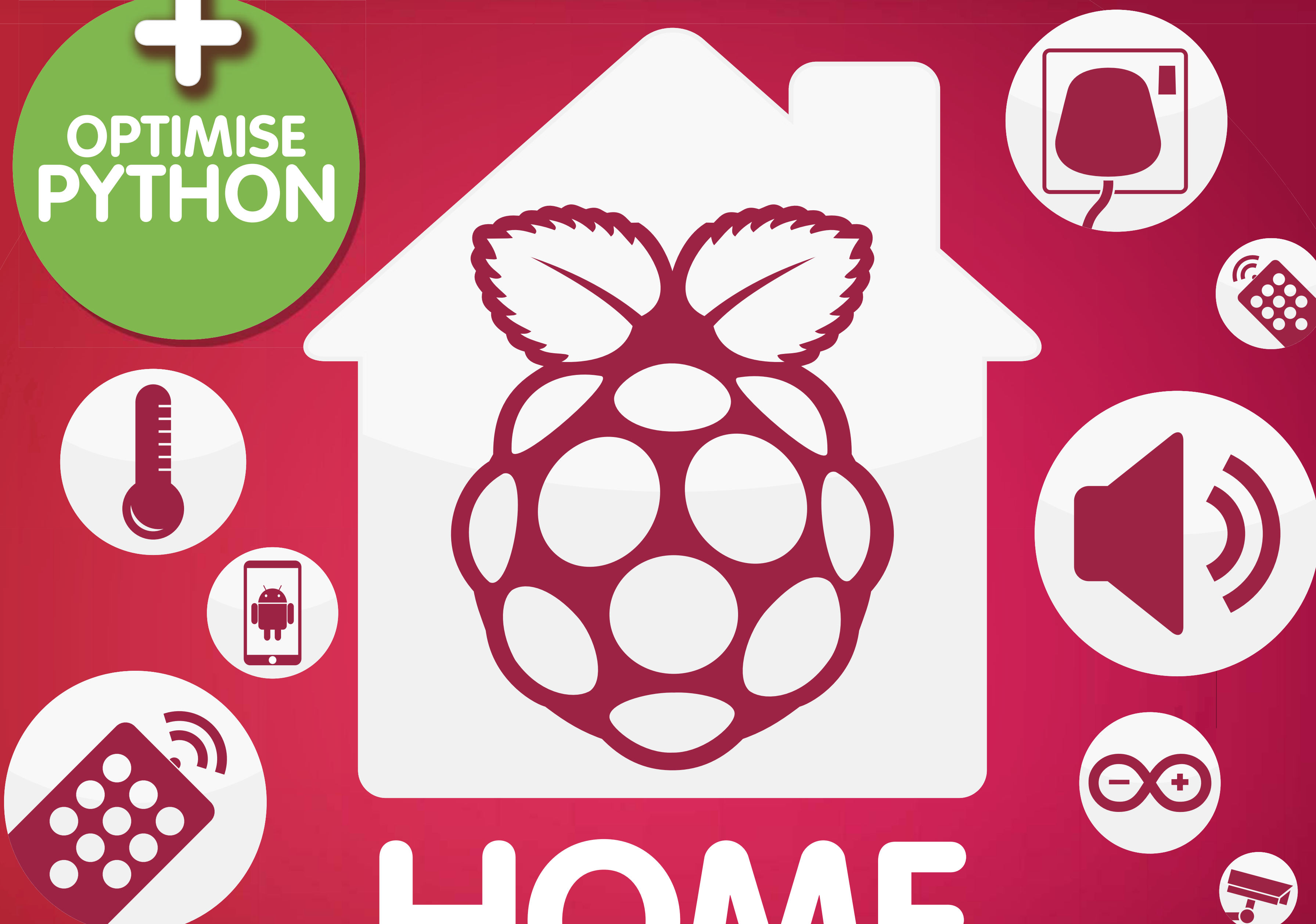
RasPi

DESIGN
BUILD
CODE

23

Get hands-on with your Raspberry Pi

+ OPTIMISE PYTHON



HOME AUTOMATION
WITH YOUR PI

Plus Build a radio transmitter

Welcome

Our homes are getting cleverer, but not everybody wants the cost (or the restrictions) of big brand-name Internet of Things devices – especially not when all you need to make your own is a Raspberry Pi, a few tips, tricks and HATs and some ingenuity. In this issue you'll learn how to make your own automated system for your home, which you can use to switch sockets, open and close your garage door, check and change the temperature, keep an eye on your security and more. Want to do a test run? Learn how some PubNub staffers visualised a smart home with a Raspberry Pi and some LEGO! You'll also learn how to build a radio transmitter, how to monitor your environment and much more.

April

Editor

Get inspired

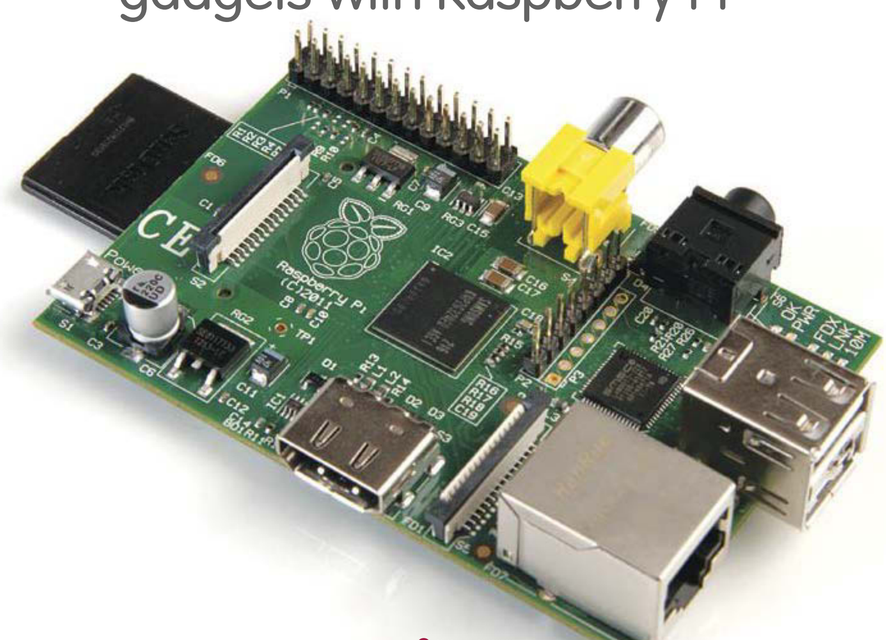
Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

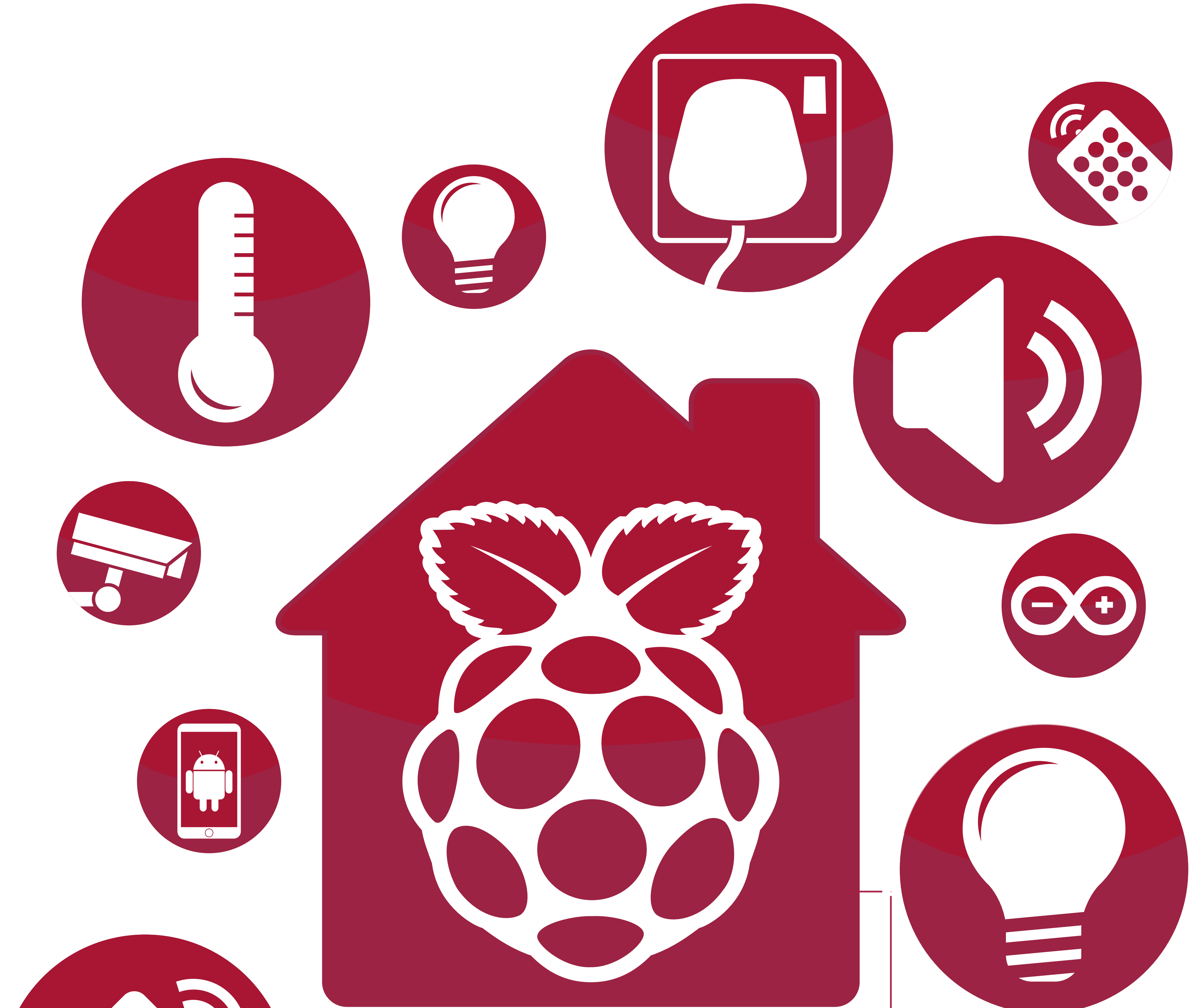


From the makers of LinuxUser & Developer

Join the conversation at…

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk

Contents

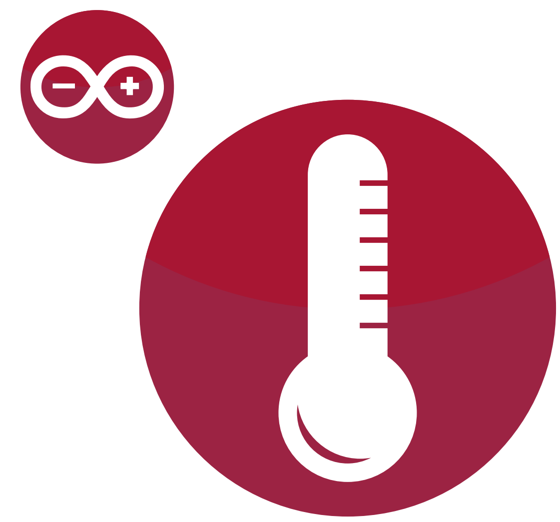


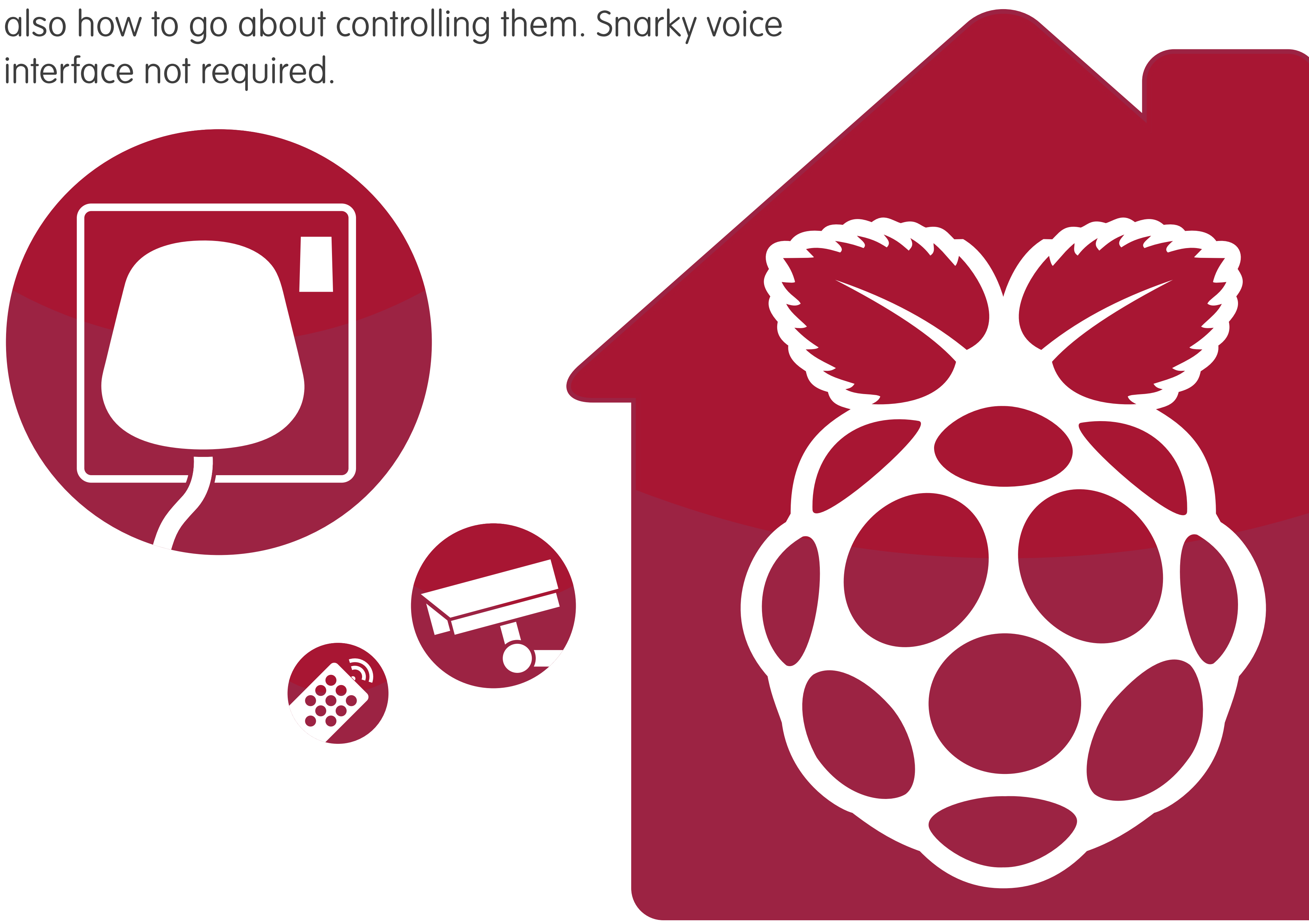
HOME AUTOMATION

WITH YOUR PI

We regularly talk about how the Raspberry Pi is the perfect little computer to have around the house doing work that requires just enough computing power to keep it running. It's not the purpose of the device, but it is just really good at it. File servers, media centres, etc – its size and flexibility make it an often surprisingly powerful tool.

And we can always go a step further. Instead of handling idle computing tasks around the house, what if we had it control the house? Through modern home automation and a bit of fancy coding, you can easily make the Raspberry Pi do a little bit more and control many aspects of your home. In this feature, we're going to run you through not only setting up the controllable lights and thermostats and such, but also how to go about controlling them. Snarky voice interface not required.





Your smart home setup

Remote control sockets

Energy saving and green houses are a big thing right now, and you can buy power strips that will shut down every socket based on the draw from a single socket. This isn't always accurate, though, and being able to manually control the socket is not always easy if it's hidden away or part of a power strip. With the use of remote control sockets, you can control the power of anything from the Pi and a web interface, enabling better control and less use of device standby modes.

Lights

A classic home automation function is controlling the lights in the house depending on the time of day or how dark it is. There are many ways you can do this: the popular method right now is Wi-Fi enabled bulbs, allowing for direct control, but you can also use the remote control sockets or use strips of LEDs that can easily light a room and are much easier to manually interface with. With all of these methods separate from the automated control, you can also remotely control the lights to switch on and off as you please. It's not a good idea to try and spook house guests, though.

Thermostat

Technologies like Nest are becoming extremely popular, but connected thermostats have been around for a long time – longer for those with a soldering iron. While we're not going to be quite creating a thermostat that 'learns', using it for external control and monitoring is easy enough when you have the right equipment. We'll be concentrating on the monitoring part in this tutorial, using a thermistor and a bit of calibration to figure out the temperature.

Security doorbell

It's surprisingly easy to get one of these home security systems set up. Using technology created for security cameras and streaming, you can create a live feed using the Raspberry Pi that can be displayed wherever you want. This can be done quite simply on the Pi using a webcam or even the Pi camera itself, and you can even try and hook it into the doorbell if you want a really cool party trick.

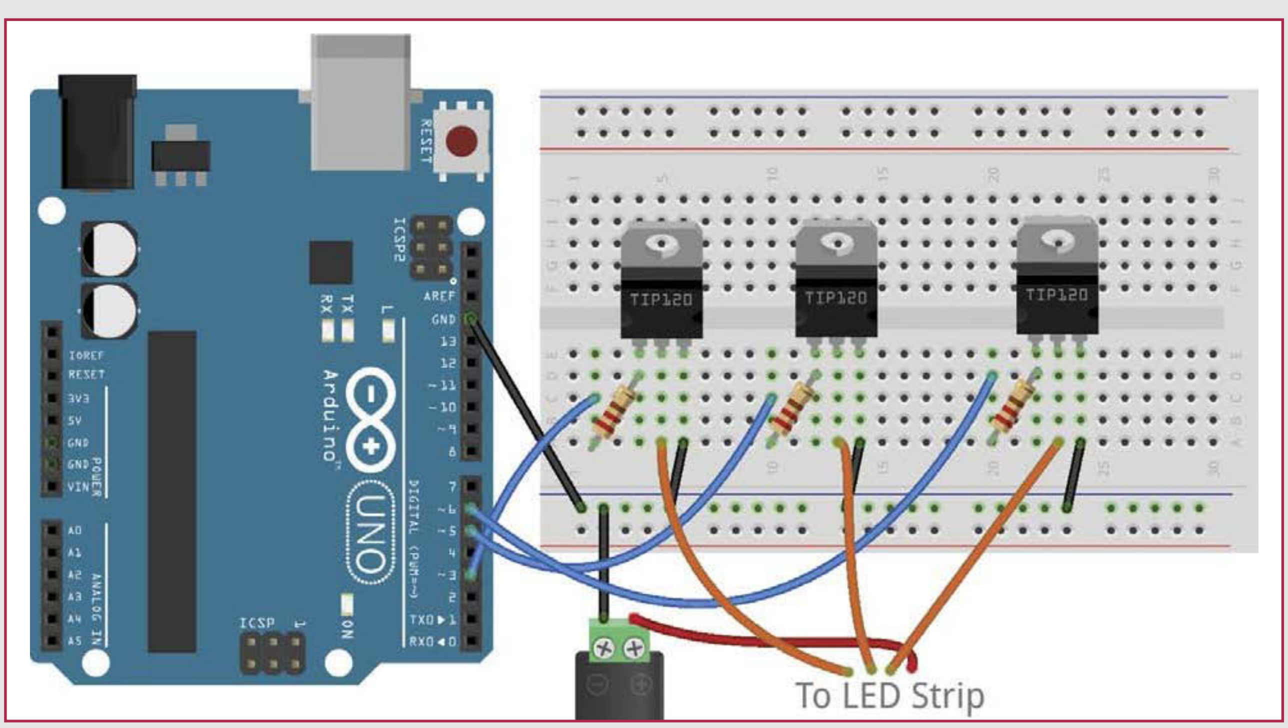
Build your own automated system

Discover how to remotely switch sockets, your garage door, control a strip of LED lights and more

Remote control sockets

We are going to use a 433MHz receiver and transmitter module (these can be found for a couple of pounds on eBay – simply search for '433MHz' and you'll find what you're looking for) connected to an Arduino to switch a pack of remote control sockets. We used a pack of four remote control sockets from Energenie (£17 on Amazon). Remote control sockets are ideal for items such as floor-standing lamps, and anything else without an on/off switch. In our expert's case, he has an audio mixer that doesn't have an on/off switch.

Once you have the sockets set up to work with the



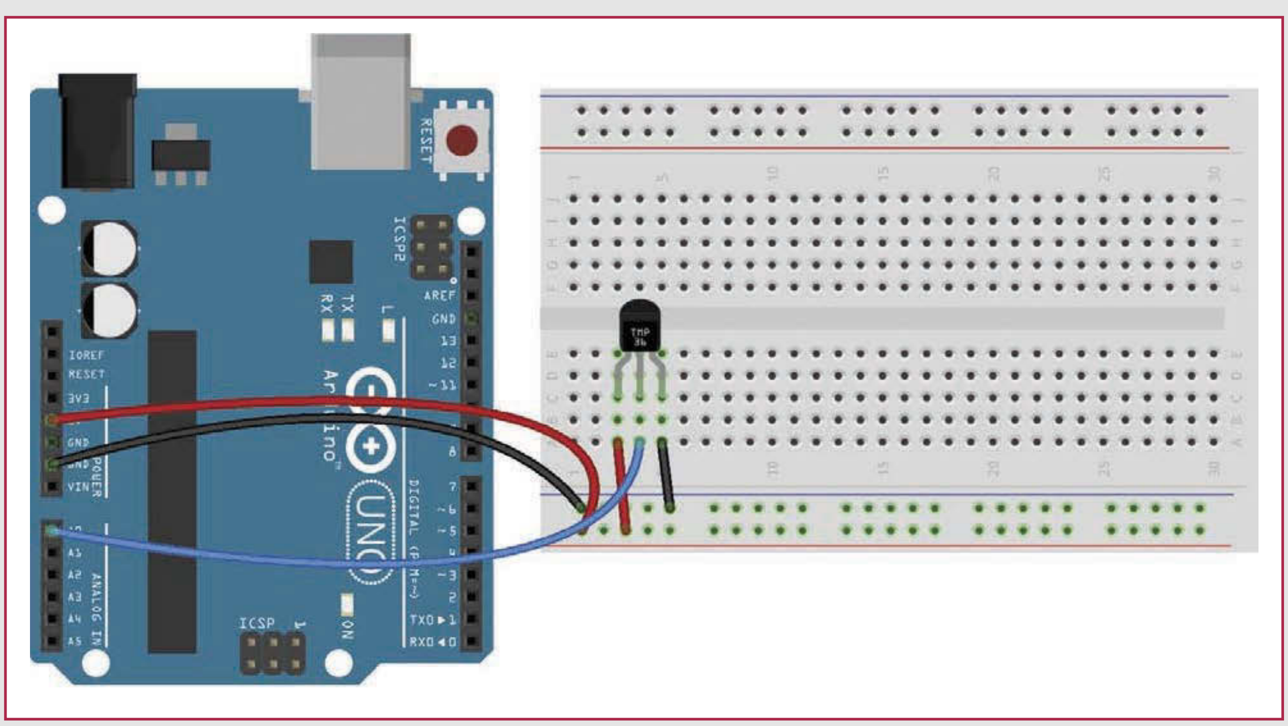
Left The three separate colours used for the RGB sequence are wired manually to the LEDs

remote, you can use the 433MHz receiver and a simple piece of Arduino software to capture the message sent by the remote so it can be sent later using the transmitter module. The modules have very simple wiring: 5V, Ground and Data. The receiver has two pins for the data line but you only need to connect the Arduino to one of them. The beauty of sniffing remote control codes is that it can be used for anything you like that works at 433MHz – remote control garage doors or light switches are likely to use 433MHz.

LED light strips

LED light strips are great. They are very easy to find on Amazon and cost about £15 for a length of five metres with a 12V power supply and remote. They have adhesive on the back so they can be stuck to a surface. Alternatively, you can just leave it on the reel as that will still give off a lot of light.

Each LED on the strip has an individual red, green and blue colour component. These colours can be set to any brightness level between 0 and 100% using pulse width modulation. This is where the signal for each colour is a square wave that is on a certain amount of the time. For example: if the Red signal had a 50% duty cycle then it would be on for 50% of the time, resulting in reduced brightness.

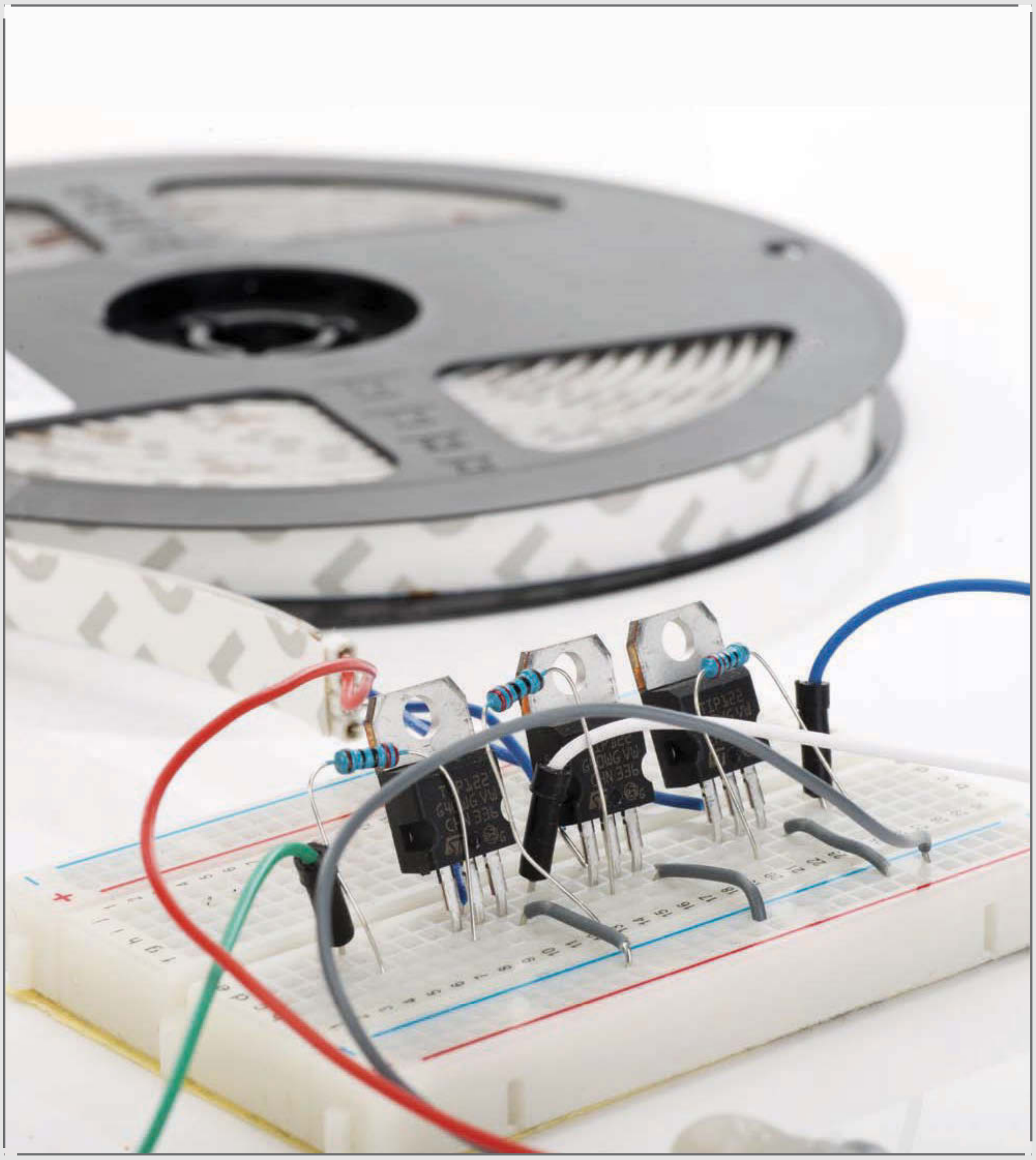


Left The temperature sensor is very simply wired up, connected to positive and ground rails along with a data pin

Left The Raspberry Pi camera plugs directly into the Raspbery Pi itself, not requiring extra wiring

The strip has four connectors on it: +12, Red, Green and Blue. The colour connectors are the ground pins for each colour. When the pin for a colour is connected to ground, the circuit is completed, allowing 12V to flow through the strip and to ground. We will use a high current transistor for each colour to control the connection. Note that the ground from the Arudino and power supply should be connected together on the breadboard. The 12V supply for the LEDs should not connect to the Arduino – only to the LED strip – or the Arduino will get damaged.

We used a TIP120 NPN transistor, which is capable of



Left Here's a better look at the colour controllers hooked up to the LED lights

switching 5 amps – this is plenty for our application (the power supply that comes with the strip only supplies 2.5 amps, and we can switch 15 amps because there is a 5 amp transistor for each colour). This transistor would also be good for controlling the speed of a fan or other kinds of motors.

The pins of the TIP120 are Base, Collector and Emitter, from left to right. The base is connected to the PWM signal from the Arduino via a 220-ohm resistor. The collector is connected to one of the colour pins of the LED strip, and the emitter is connected to ground. Note that when passing high amounts of current, these chips get hot. Also, it is wise to use

solid core wire from the emitter-to-ground and collector-to-LED strip wires because they can safely handle more current than breadboard jumper wires. You need to ensure that none of the wires connected to the strip can short, otherwise you could damage the LEDs in the strip. Our setup here is only temporary – it could be worth moving the circuit onto veroboard so that it is more stable once you have tested that it works on a breadboard.

Temperature sensor

The TMP36 is a very simple IC. It has three pins: 5V supply, ground and a voltage out from 0-2V to represent temperature. This variable voltage can be read with the analogue in pins of an Arduino. The formula is: Temp in °C = [(Vout in mV) – 500] / 10, so a voltage of 0.7V would be 20°C.

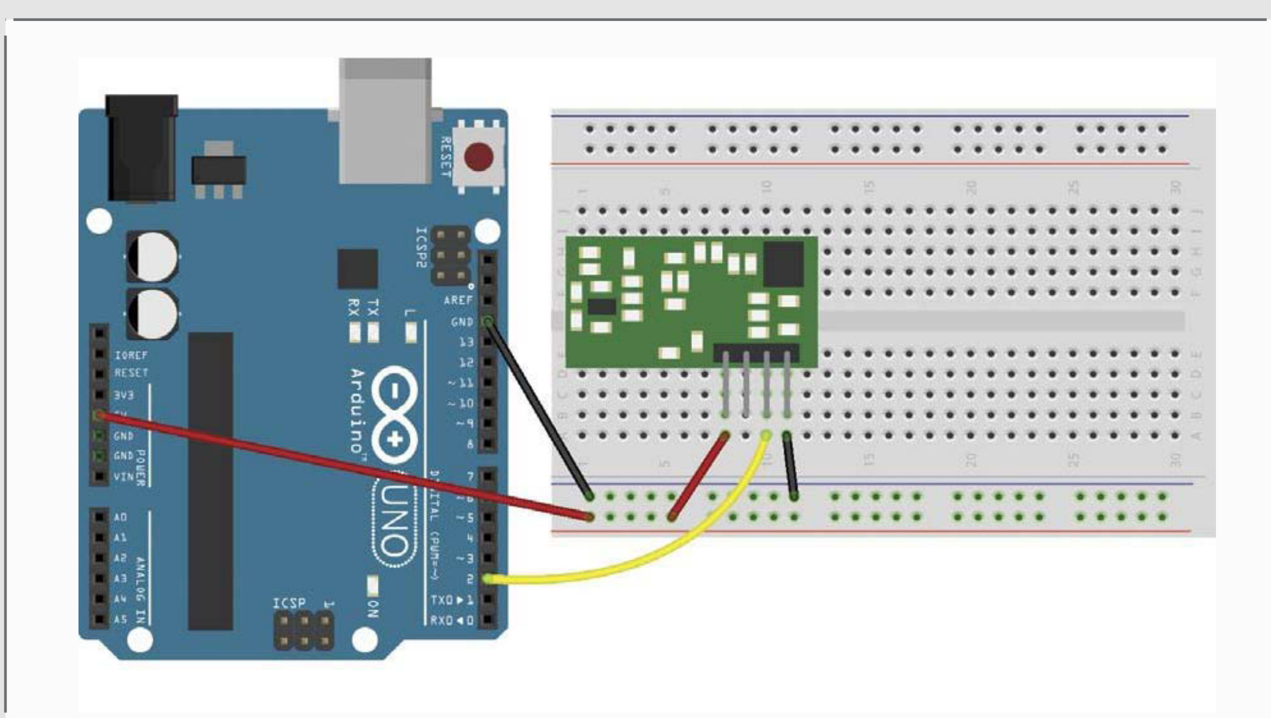
Camera

You can either use a USB webcam or Raspberry Pi camera for the video stream. The Raspberry Pi camera should be connected with the blue plastic facing the Ethernet connector, and the exposed traces of the ribbon cable facing towards the HDMI connector.

Control your automated system

Configure your Raspberry Pi and Arduino to work in harmony for an automated home

We are going to use heimcontrol.js as the control software for our automated system. This is a home automation web interface written in Node.js that runs on the Raspberry Pi, and sends low level serial commands to an Arduino connected via USB to control various hardware connected to it. Although some of the things the Arduino does can be done with the Raspberry Pi in theory, the Raspberry Pi does not have the ability to read analogue voltages, so temperature and light sensors would not be possible in this case. Also, the Raspberry Pi only has one hardware pulse width modulation output and three are needed for the LED strip.



Left The radio receiver needs power for it to work and then a connection to the data pin

Raspberry Pi prep

Use the latest Raspbian image as a starting point for this project. Log into the Pi using the default username of "pi" and default password "raspberry". The first thing you'll need to do is use **sudo raspi-config** to enable the camera and resize the root filesystem so the image uses all of the available space on the SD card. Once this is done, we can update our packages to the latest version and then install some extra packages that will be required later on:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install libboost-dev arduino
   streamer
```

Next, we need to download a precompiled Node.js package and a MongoDB package (both are required by heimcontrol.js. Packages can be found at **http://liamfraser.co.uk/lud/homeautomation**, in case they go missing.

```
wget https://node-arm.herokuapp.com/
   node_0.10.36_armhf.deb
sudo dpkg -i node_0.10.36_armhf.deb
```

Check it has installed correctly:

```
pi@raspberrypi ~ $ node -v
v0.10.36
```

Time to install MongoDB…

```
wget https://github.com/tjanson/mongodb-armhf-
    deb/
releases/download/v2.1.1-1/mongodb_2.1.1_armhf.
    deb
sudo dpkg -i mongodb_2.1.1_armhf.deb
# Start service:
sudo /etc/init.d/mongodb start
# Automatically start service at system
    startup:
sudo update-rc.d mongodb defaults
```

Now it's time to install heimcontrol.js, which our expert had to fork on GitHub to fix a couple of issues.

```
npm config set python python2.7
git clone https://github.com/liamfraser/
heimcontrol.js.git
cd heimcontrol.js
npm install
```

The install process will take a while as there's quite a lot of stuff to compile. Before you can access heimcontrol.js, you will need to know the IP address of your Raspberry Pi. You can find this out using the ip addr command. Our address was 172.17.173.41. Run heimcontrol.js by typing:

```
node heimcontrol.js
```

Note that you have to be in the directory where you cloned the repository for this to work. This is probably /home/pi/heimcontrol.js. Heimcontrol runs on port 8080, so type the IP address of your Pi into your web browser followed by :8080

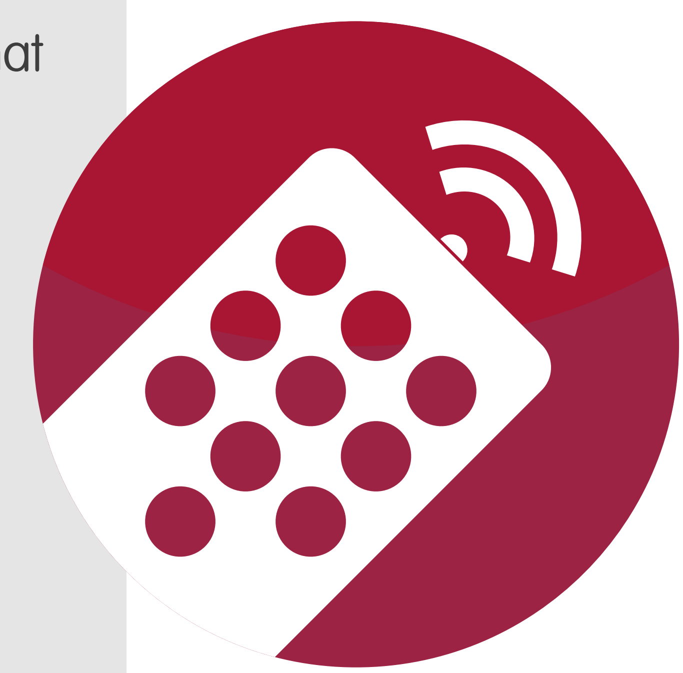
– in our case the correct URL was: http://172.17.173.41:8080. We have applied a patch that disables authentication by default, because it gets annoying if you have to log in every time you want to use the web interface. You can re-enable authentication by editing config/development.js inside the heimcontrol.js directory.

Now that we know heimcontrol is working, Ctrl+C out of it because we have more work to do before we can start using it. We need to load the video for the Linux camera driver for the Raspberry Pi camera so that it can be used with the streamer software we installed earlier. To do this, you need to edit /etc/modules using sudo and your favourite text editor (use nano if in doubt, so sudo nano /etc/modules). Add the line "bcm2835-v4l2" to the end of the file so the driver is loaded at boot. To load it instantly, run sudo modprobe bcm2835-v4l2.

Arduino prep and remote control scanning

We need to write some software to the Arduino called duino, which allows the ports to be controlled over serial from heimcontrol.js. This way of working is elegant because it allows more sensors to be added to the Arduino without any need to reprogram anything. We have already installed the Arduino software, so now we need to copy some libraries required by duino to the Arduino installation directory so that the software can be compiled:

```
cd /usr/share/arduino/libraries
sudo cp -r /home/pi/heimcontrol.js/node_
   modules/
duino/src/libs/* .
cd ~
```



Before we write the duino software to the Arduino, we want to use the Arduino to sniff the messages sent by the remote for the remote control sockets. To do this, connect the 433MHz receiver module (the wider module of the two modules with four pins instead of three – see diagram to left) to the Arduino. Connect the data pin (either of the middle pins) to pin 2 of the Arduino, VCC to 5V, and GND to GND. Download the receiver software using:

```
wget https://raw.githubusercontent.com/sui77/
rc-switch/master/examples/ReceiveDemo_Simple/
ReceiveDemo_Simple.pde
```

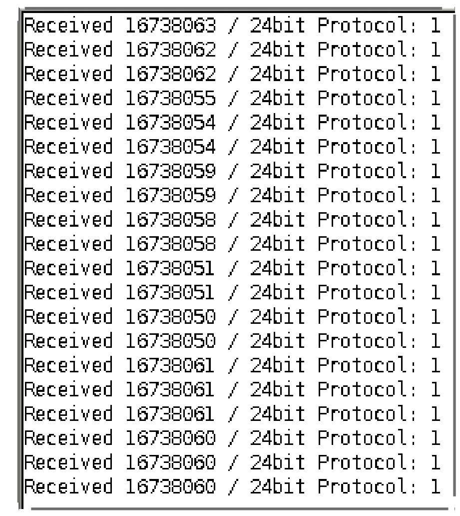
… which is again mirrored over at **http://liamfraser.co.uk/lud/homeautomation/ReceiveDemo_Simple.pde**.

Now we have to start the Arduino software. If you are connecting to the Pi via SSH then you can enable X11 forwarding to your local machine by logging in with:

```
ssh -X pi@172.17.173.41
```

Alternatively, you can type startx to start an x session if you have a screen and keyboard connected to the Pi. Once you have logged in with your desired method, type arduino into a terminal to start the Arduino programming software.

Open the ReceiveDemo_Simple.pde file that you just downloaded and upload it to the Arduino. Then open the serial monitor (Tools>Serial Monitor) and press the reset button on the Arduino. By pressing each button on your remote, you can see the code to switch each socket on and off. Make a note of the codes for each button because you will need to enter them later. Our output can be seen in the top-right image.

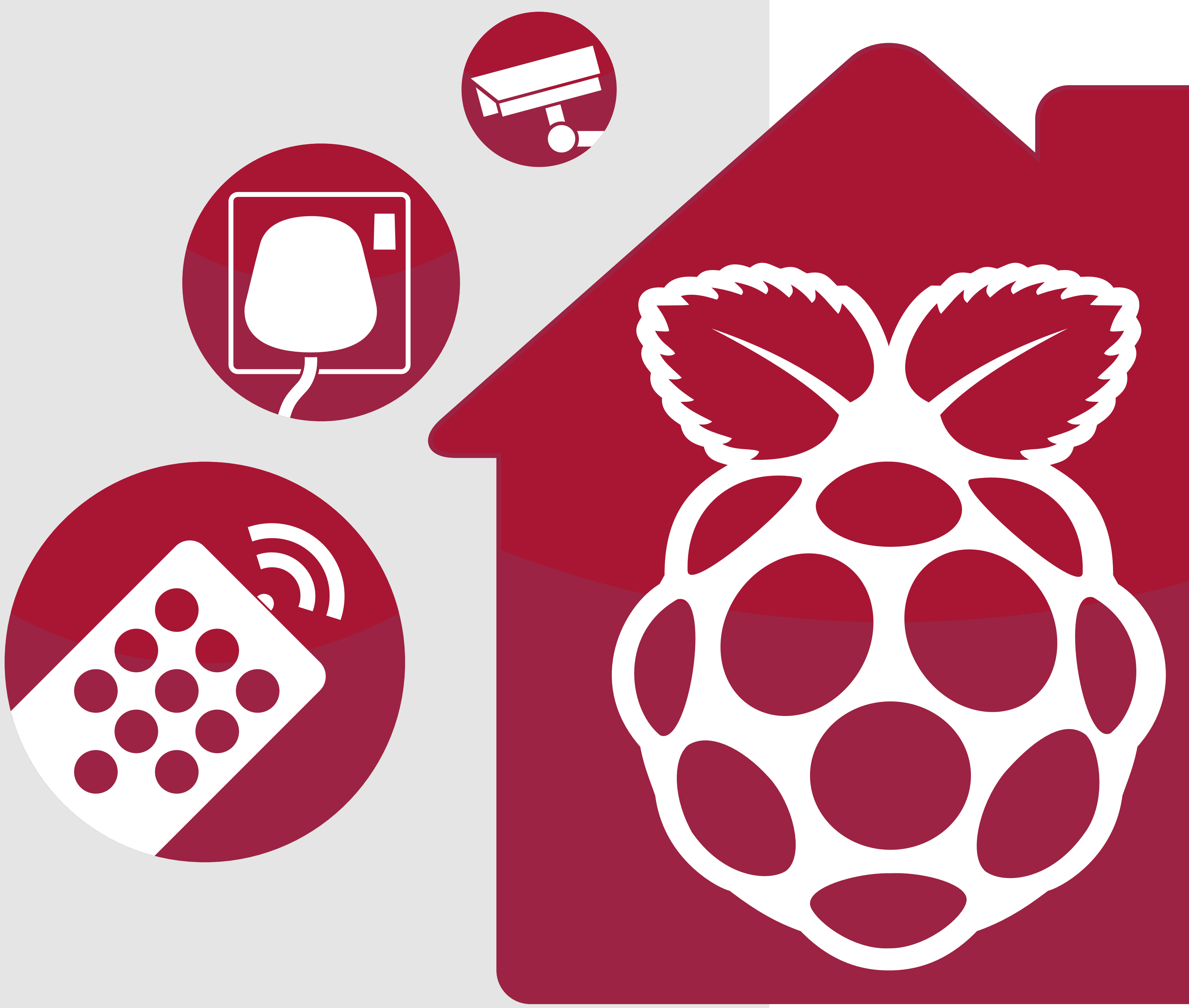


```
Received 16738063 / 24bit Protocol: 1
Received 16738062 / 24bit Protocol: 1
Received 16738062 / 24bit Protocol: 1
Received 16738055 / 24bit Protocol: 1
Received 16738054 / 24bit Protocol: 1
Received 16738054 / 24bit Protocol: 1
Received 16738059 / 24bit Protocol: 1
Received 16738059 / 24bit Protocol: 1
Received 16738058 / 24bit Protocol: 1
Received 16738058 / 24bit Protocol: 1
Received 16738051 / 24bit Protocol: 1
Received 16738051 / 24bit Protocol: 1
Received 16738050 / 24bit Protocol: 1
Received 16738050 / 24bit Protocol: 1
Received 16738061 / 24bit Protocol: 1
Received 16738061 / 24bit Protocol: 1
Received 16738061 / 24bit Protocol: 1
Received 16738060 / 24bit Protocol: 1
Received 16738060 / 24bit Protocol: 1
Received 16738060 / 24bit Protocol: 1
```

Above Here are the codes that we sniffed from our remote control socket controller

Once this is done, we can finally write the duino software to the Arduino. This process is the same as what you've just done except the file is located at /home/pi heimcontrol.js/ node_modules/duino/src/duino/duino.ino.

The software might take a minute to compile. Once it has been uploaded, you can exit the Arduino software and press the reset button on the Arduino. Now we can put everything together and start adding our sensors to heimcontrol.js.



Set up your control interface

Now is the time to start adding our sensors and devices to heimcontrol.js

01 Start heimcontrol.js on boot

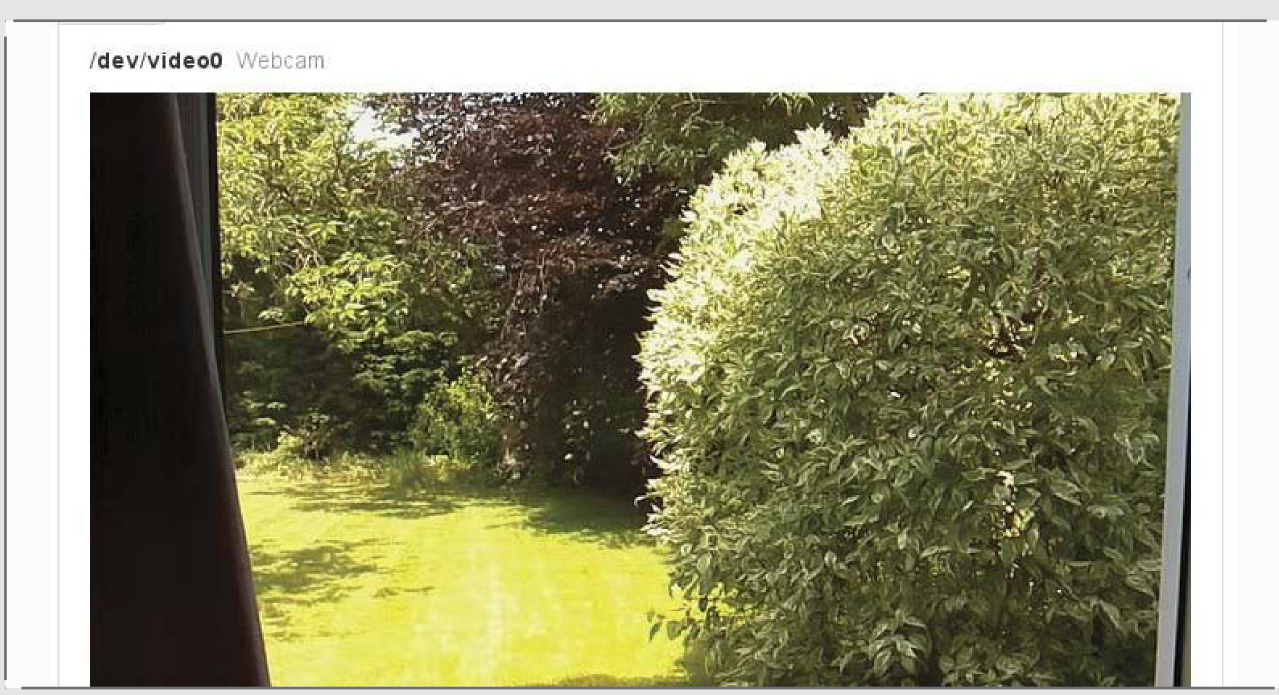
Before we start adding devices, it makes sense to start heimcontrol.js on boot. To do this, we can add a line to /etc/rc.local which is a script that gets ran at boot by the root user. The file needs to be edited with sudo and your favourite editor, for example:

```
sudo nano /etc/rc.local
```

Add the following line before the line "exit 0":

```
su pi -c "node /home/pi/heimcontrol.js/
    heimcontrol.js" &
```

Heimcontrol.js will be started automatically at boot from now on, but for now you can start it with node /home/pi/heimcontrol.js/heimcontrol.js.



Left Set up a camera to monitor your front door, or to keep an eye on the garden

02 Add the camera feed

Go to Settings and select Webcam. Set the method to Streamer and the devices as /dev/video0. Pick an interval; we picked two seconds but you can pick any interval you like. Shorter intervals are more feasible on a Raspberry Pi 2, as it is generally more responsive. Click Save and then go back to the home page.

03 Prepare remote control socket codes

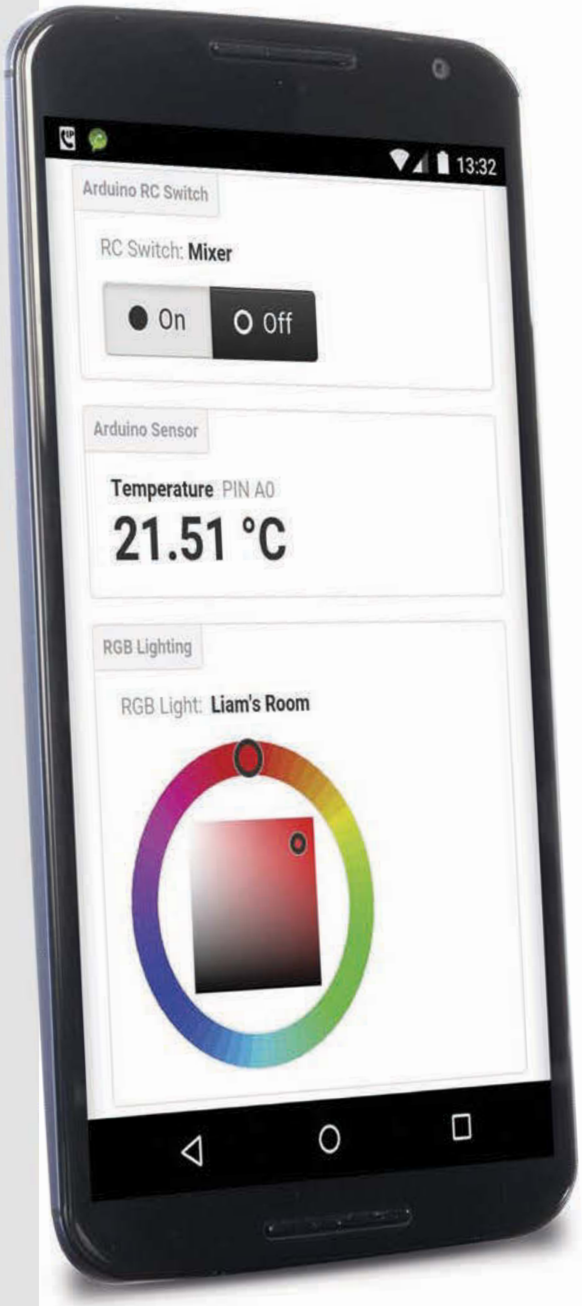
This is where you need the codes that you sniffed from the remote earlier on. The heimcontrol.js web interface takes the code as a binary string of 1s and 0s. However, the code we sniffed is in a different format so you'll need to convert it to binary using Python. Type python2 into the terminal to open a Python interpreter. Format the integer you captured as a binary string like so:

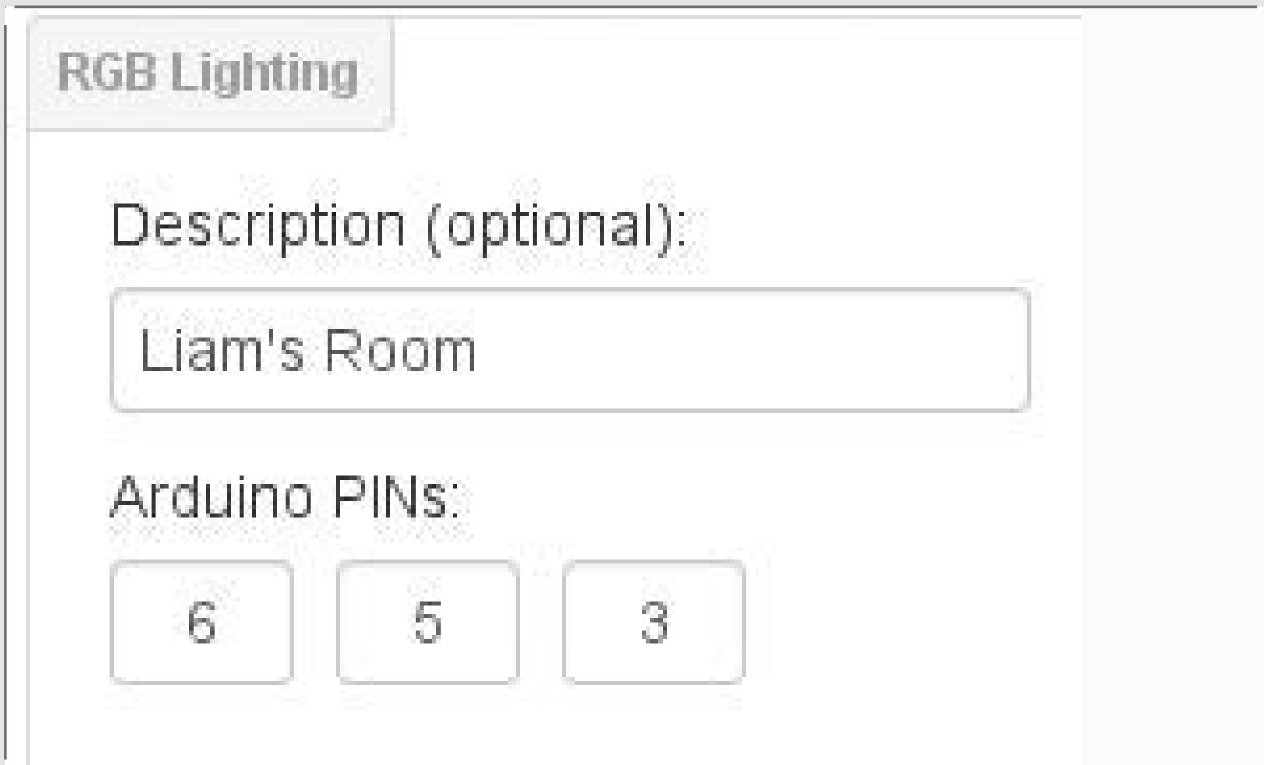
```
>>> "{0:b}".format(16738063)
'111111110110011100001111'
```

04 Add the remote control socket

Go to the Settings menu and go to the Arduino section. Click the Add button and set the method to RC Switch. Set the code type to binary. Give the switch a name, enter the pin that the RF transmitter is connected to (in our case, pin 2) and enter the two codes that you just worked out for the on/off buttons. Go back to the home page and test that the switch works. If it doesn't, you might need to add an antenna to the transmitter by making a loop of wire. Check everything is connected correctly.

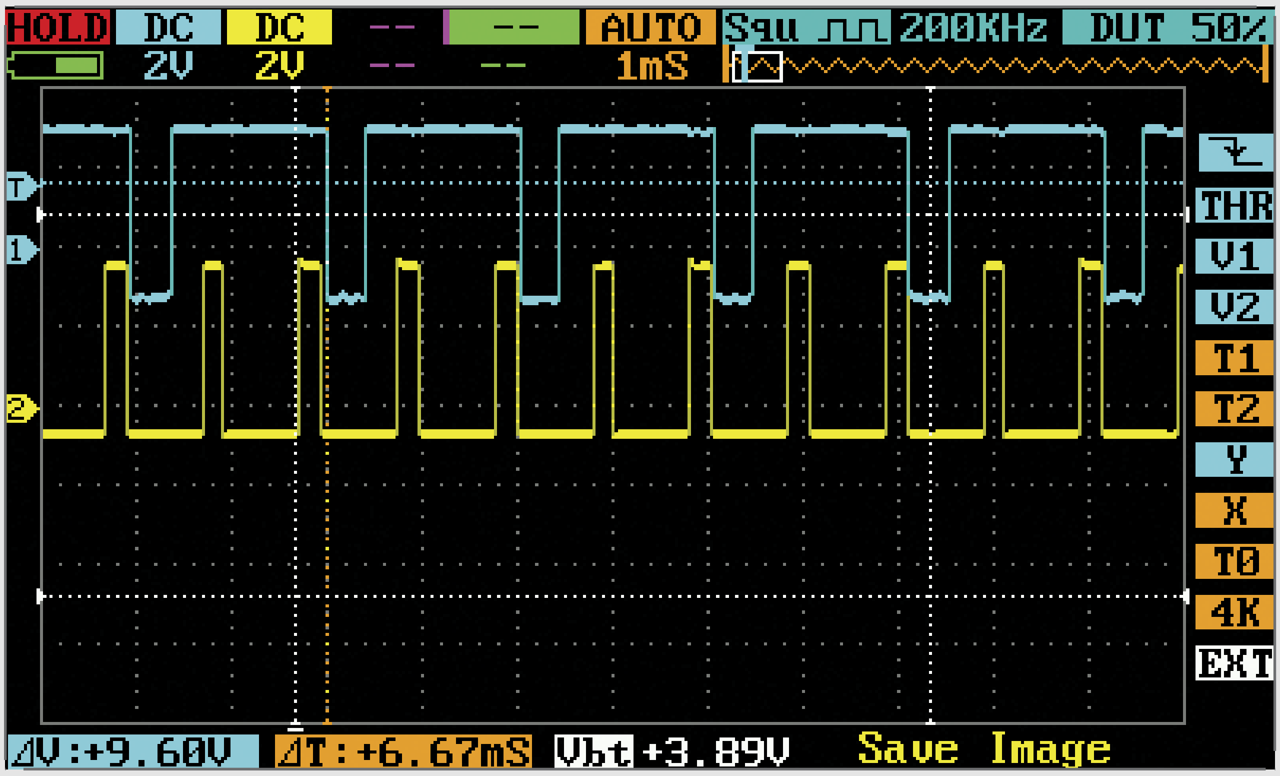
Below The heimcontrol.js interface works really well on mobile devices





05 Add the temperature sensor

The temperature sensor can be tricky because it needs calibrating. A multimeter is a good idea so you can accurately read the analogue voltage. Go to the Arduino settings page and add a sensor. The formula for the TMP36 is: [(Vout in mV) - 500] / 10. We read 718mV with a multimeter, which would put the temperature at 21.8°C. Experiment with the formula by seeing what the raw value of x is from your sensor, but ours ended up as: (((x+45) * (5000/1023.0)) - 500) / 10. (5000/1023 is because the Arduino has a 10-bit analogue-to-digital converter, ie 0-1023 to read a voltage up to 5V.) Note that you have to ensure you have perfectly matched brackets, otherwise the software will crash because it simply tries to evaluate the string you put in.



Left The pulse width modulation signals from the Arduino. One wave is on 20% of the time, the other 80%. This controls the brightness of each colour to create any colour from the three primary colours

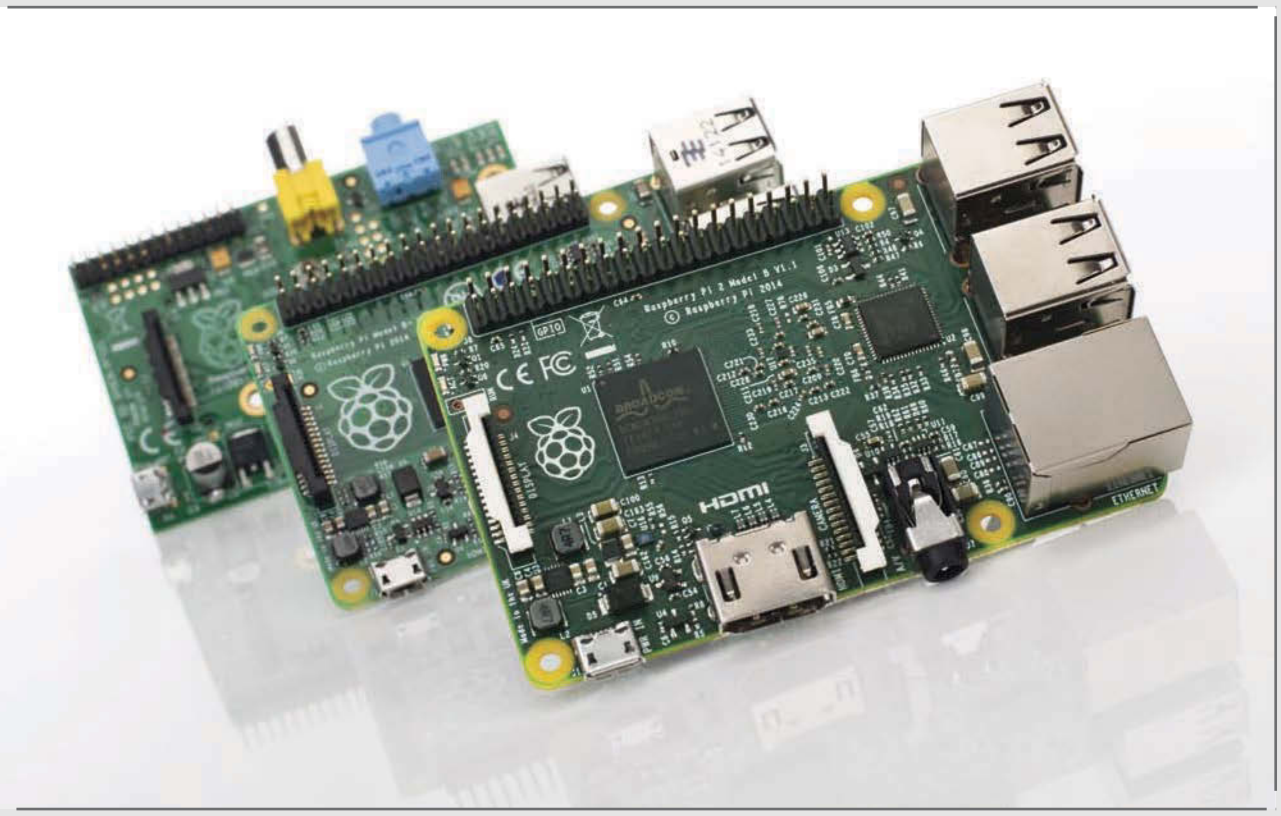
06 Add the LED strip

Finally, it's time to add the LED strip. Make sure the signal wires are connected to PWM-capable pins on the Arduino marked with a ~. We used pins 3, 5 and 6. Go to Settings>RGB Lights. Ensure you have the signal wires connected to the correct colours. We used 6 for red, 5 for green and 3 for blue. Click Save and turn on the 12V supply. Select a colour from the colour picker. Your strip should light up! Pick Red, Blue and Green to ensure everything is wired up correctly.

Expand into the future

Set up multiple systems

Now that you have heimcontrol.js set up in one room, you could extend the system by setting up multiple Raspberry Pis and have one in each room of the house – potentially a good way of using all of the older models that are gathering dust in your drawers. You could either have a master Raspberry Pi that reverse proxies multiple instances of the web interface depending on the link you give it: /livingroom, /kitchen and so on, or just simply have a bookmark for each room. If you were eager to have as few cables as possible, you could always look at getting a power-over-Ethernet injector/splitter that could send 5V power over an Ethernet cable along with the network connection.



Left You could have one master Raspberry Pi or individual ones in each room

Set up an audio system

A nice addition to home automation would be some kind of multi-room audio system. If there's already a Raspberry Pi in each room, you only need a cheap class-T audio amplifier (£20), a USB sound card for better audio quality and some bookshelf speakers to get this going. A pair of powered PC speakers would also do the trick. If you want just one set of speakers then mopidy (a music player daemon with Spotify support and a web interface) would be fine. Alternatively, you could look into setting up Squeezebox, which is multi-room audio software that was originally developed by Logitech but is now open source.

LEGO Smart Home

Bhavana Srinivas and Geremy Cohen from PubNub show us how you can connect together Internet of Things devices as easily as blocks of LEGO

So tell us about PubNub and your LEGO Smart Home.

PubNub is a global data stream network. What this means is that, using the PubNub software, you can talk between devices in real-time. And, going beyond that, we provide an infrastructure as well, so you can scale and build real-time applications. The simplest case is when you have two mobile phones talking to each other – using any kind of instant messaging or a commercial app that requires stocks to be updated in real-time, for example, or any kind of home automation – that kind of real-time messaging is brought to you PubNub. Since we have 14 data centres all over the world, your data gets replicated and we're able to communicate between devices in less than a quarter of a second.

The Internet of Things was picking up at PubNub last year, so we decided to have some cool demos to show how you can integrate PubNub as a software for any kind of home automation that you build out. We were building two projects at the same time – one with the Arduino Uno (http://bit.ly/11SybiR) and the other with the Raspberry Pi, both to show home automation itself. So with this project, the Raspberry Pi Model B+ would be the brain behind the whole project, powering the different embedded sensors, the stepper motor to control the door, and we used PubNub as a glue to talk between these different devices.

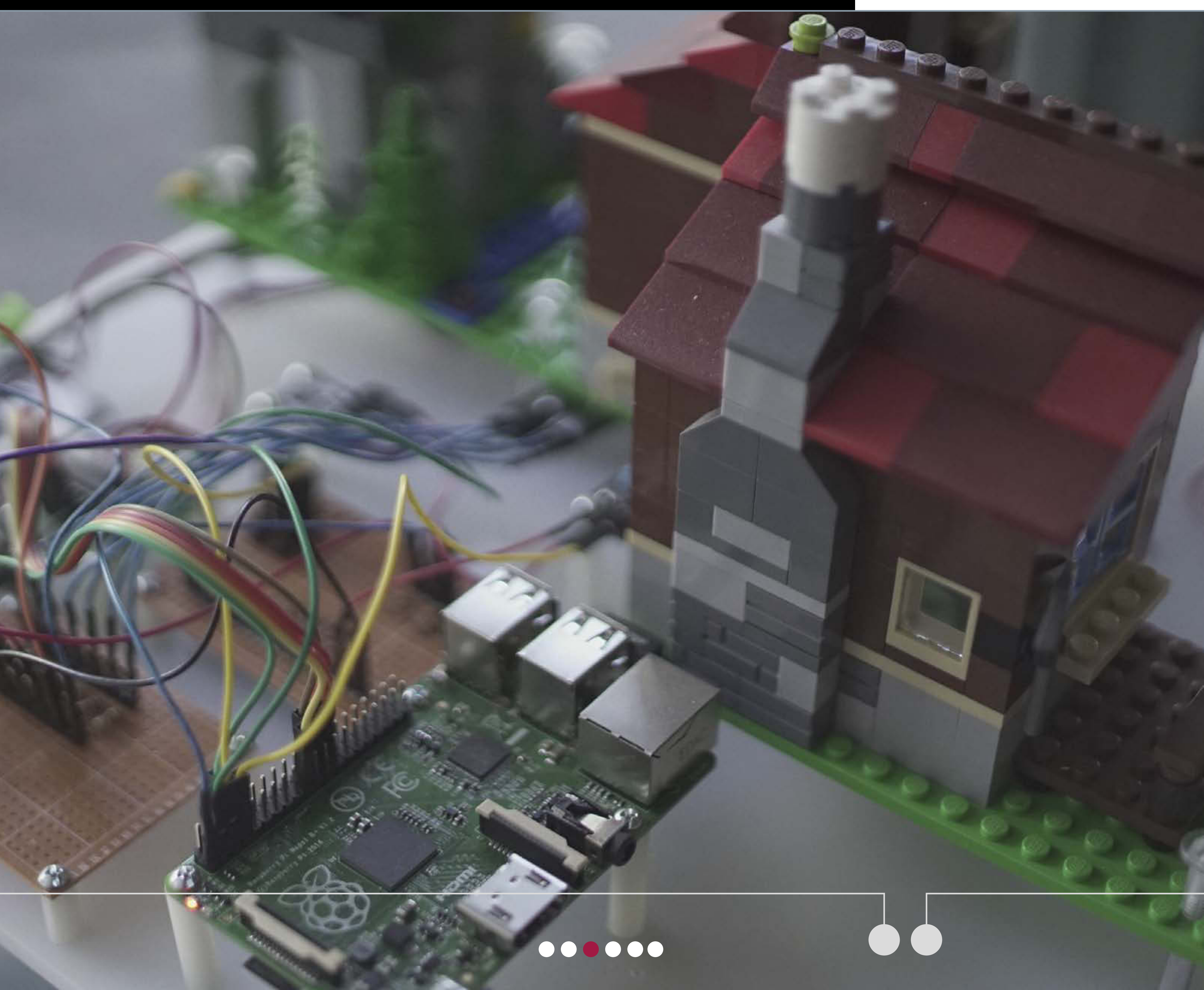
I didn't build this myself – it was actually another engineer here called Geremy who built it over Christmas. We were going to have the software component, which is PubNub, and then on a mobile phone you could control all the devices in your house; the devices being the seven embedded LEDs – used

Geremy Cohen is director of client-side engineering at PubNub, where he manages a team of brilliant engineers. His first hardware project involved repurposing a Wico joystick into a shoebox-based flight yoke to play Sublogic Flight Simulator on C64 with more gusto.

to simulate things like a stove, a fireplace, a barbecue station and lights within the house – and then we had a stepper motor to open and close the door, and a couple of sensors like temperature, humidity, etc, to measure those different values within the house.

So what you can do with this demo is open this dashboard on your phone, typically an app, and the moment you instruct it to switch on the light or open the door, it will reflect immediately on the house. You can get the state of the house continuously, in terms of the voltage or the temperature, the pressure, so that's a typical home automation system.

Below In principle, this is basically the setup you'd need for an automation setup for your own house



How does messaging between different PubNub-connected devices work?

So PubNub provides real-time communication between any two devices. What happens is that one device is publishing on a particular channel and then another device is subscribing to that same channel, and that's how the message is transmitted from the first device to the second. All you need with PubNub is the publisher key, the subscriber key and then the channel name, and when two devices are using the same set of parameters – the same pub and sub keys – and when one publisher is on that channel, the other receives it on the same channel. On a very basic level, that's how PubNub works.

How much of the software setup was specific to this project – was there a lot of work involved to get PubNub talking to the Pi?

So how it works with PubNub is that we have 70+ SDKs, which means we support that many platforms or devices – so if you're using an iOS device then you can use our iOS SDK, or there's our Python and Java SDKs, and so on. For the Raspberry Pi we chose to use our Python SDK. PubNub provides very easy-to-use APIs, so if you wanted to send out a message then it would be as simple as **pubnub.publish** and to receive a message it would be **pubnub.subscribe**, so the SDKs are already built out, but there was a little bit of tweaking for it to fit this particular home automation model.

Geremy had to build out two parts: the phone or browser app, which is the dashboard that you see, and then the Python scripts running on the Raspberry Pi itself. So for the dashboard he typically used

JavaScript and then Android or iOS, depending on the phone application, and then for the Raspberry Pi itself he used Python – Jeremy just had to write a couple of scripts that basically said, 'When I receive a JSON message from my phone saying to switch on LED 1, do so at source'. So the logic for the home automation had to be written in Python, but the software and the documentation already exists.

How easy would it be for people to recreate your LEGO Smart Home model for themselves?

It's super easy. We are actually creating tutorials as well – we're done with the temperature sensor, the humidity sensor, etc, and we just have a little bit of tweaking to

Left The LEGO version allows you to see a miniaturised version of a Pi-connected smarthome

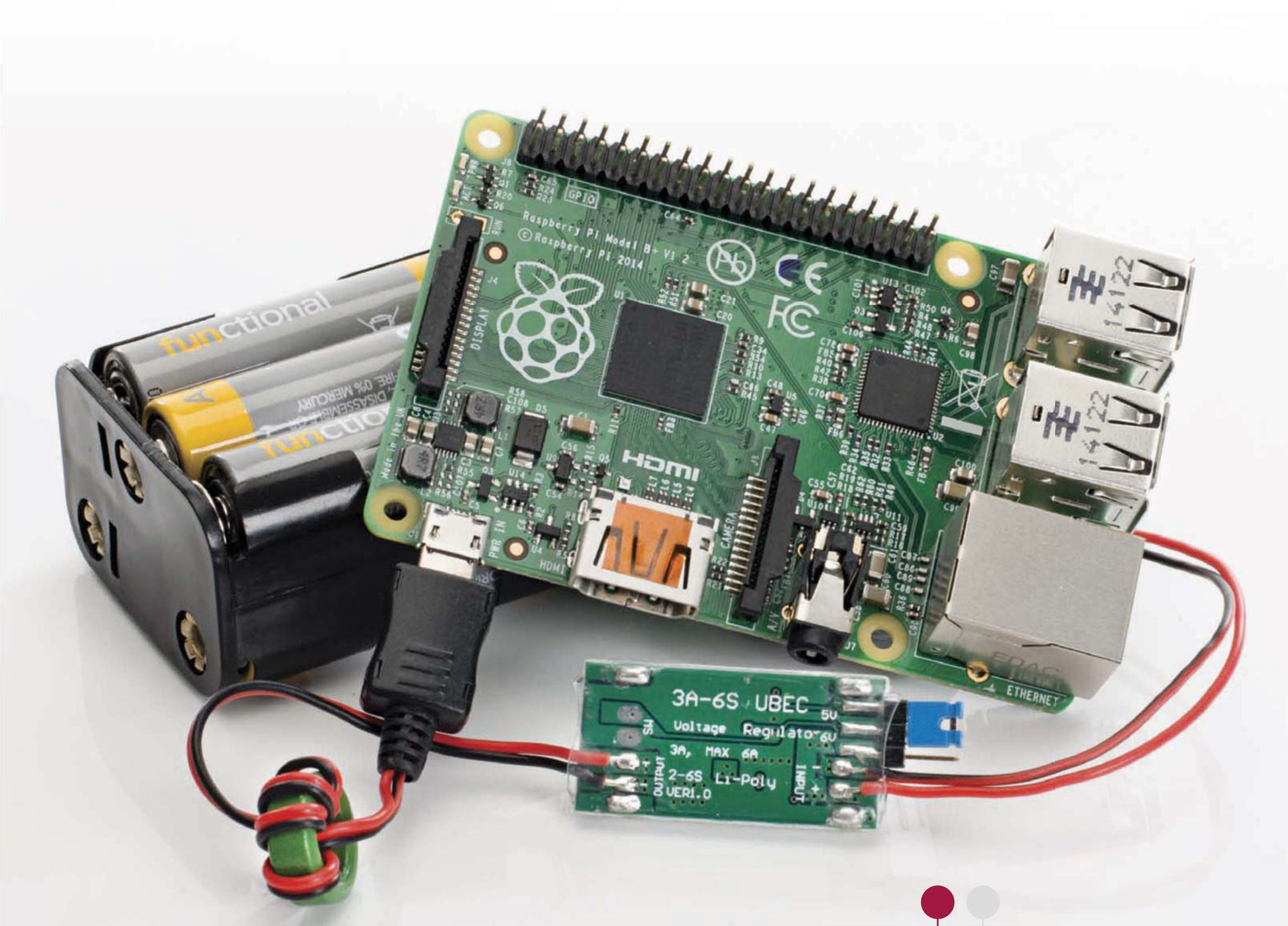
do for the lights – so they're going to be smaller follow-ups to the same video and blog that we've posted already. So it's super-easy to create and is meant for the hacker community, like a weekend or holiday project where you can just sit and build this whole thing in a couple of days. There's a free tier in PubNub where we provide a sandbox account you can use for this, wherein you're given publish and subscribe keys and you can go crazy. There's a limit on the number of messages you can send but that's way more than what you would need for a free tier. Everything's open, so you can see it on our website, and it's free up to 20 devices.

And how straightforward would it be to scale up this project and use it for real home automation, with things like Philips Hues and Sonos speakers?

We have an intern who recently built out the Hue light bulbs with the Raspberry Pi in about half an hour; it wasn't a big deal at all. PubNub is just plug-and-play – you build out all this software, put it on your hardware, and you don't have to worry about which network you're on, you don't have to configure the routers or firewalls, any of that stuff. It's very easy to scale with PubNub itself because we're global – even if you get up to millions of users, irrespective of where you are in the world you still receive the message in less than a quarter of a second.

Build a radio transmitter

Take advantage of the interference-blocking feature and make your mark on the airwaves



Back in the 1960s, offshore boats were used to broadcast what was then known as 'pirate' radio: unlicensed broadcasts that provided an alternative to the BBC's Light Programme (as the most popular radio station was then known). Pirate radio was a revolution that inspired Radio 1 and commercial broadcasting, but these days you don't need a boat to pursue your radio DJ dream – just a Raspberry Pi.

Add a basic DIY antenna, an SD card with some MP3 tunes saved to it, plus a script to automate playback, and you can follow in the footsteps of John Peel and Terry Wogan.

This is a 50-50 project, one that has a chunk of DIY as well as the usual SD card flashing. You'll also need a battery pack.

One word of warning: unlicensed broadcasting on the FM band is an offense in many countries and you need to check your local laws. This tutorial is merely a proof of concept – one that might be used for a school radio project, for instance.

THE PROJECT ESSENTIALS

Jumper wire

2mm wire

Heat shrink tubing

Soldering iron

Wire cutters/strippers

Hair dryer/heat gun

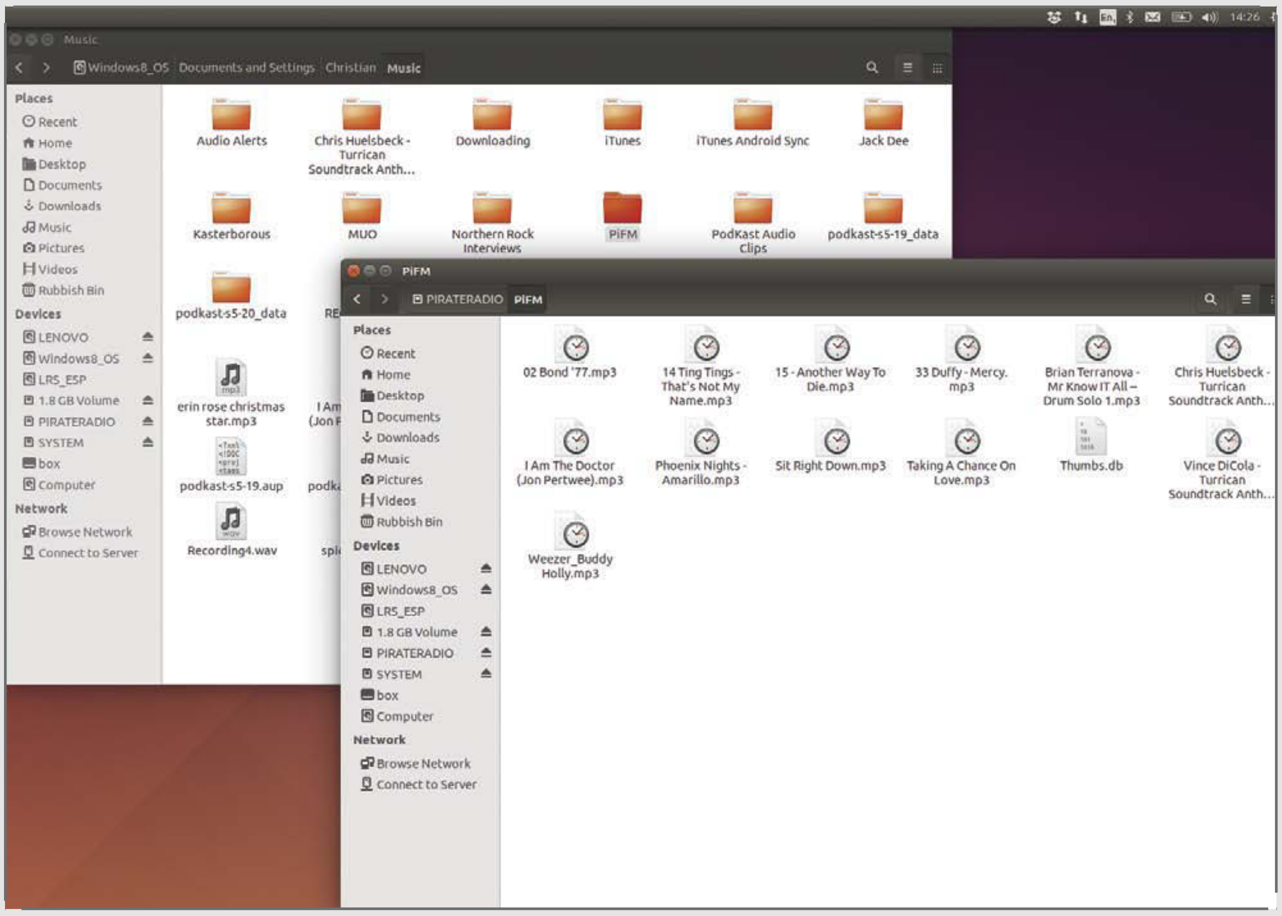
PiRadio
http://bit.ly/1MWkxwp

PirateRadio.py script
http://bit.ly/1SkkeCh

01 Gather your equipment

Begin by downloading the PiFM image. This is over 2.3GB, so if you're on a slower connection then you'll need to get it downloaded in advance.

Meanwhile, source an antenna. This might be a wire coat hanger or 2mm wire from your local electronic component store. While you're there, grab some heat shrink tubing and some jumper cables, ensuring that you're well prepared to start the project.



Left Prepare MP3 files in advance or make your own using Audacity, or a similar open source audio production tool

02 Prepare MP3 files in advance

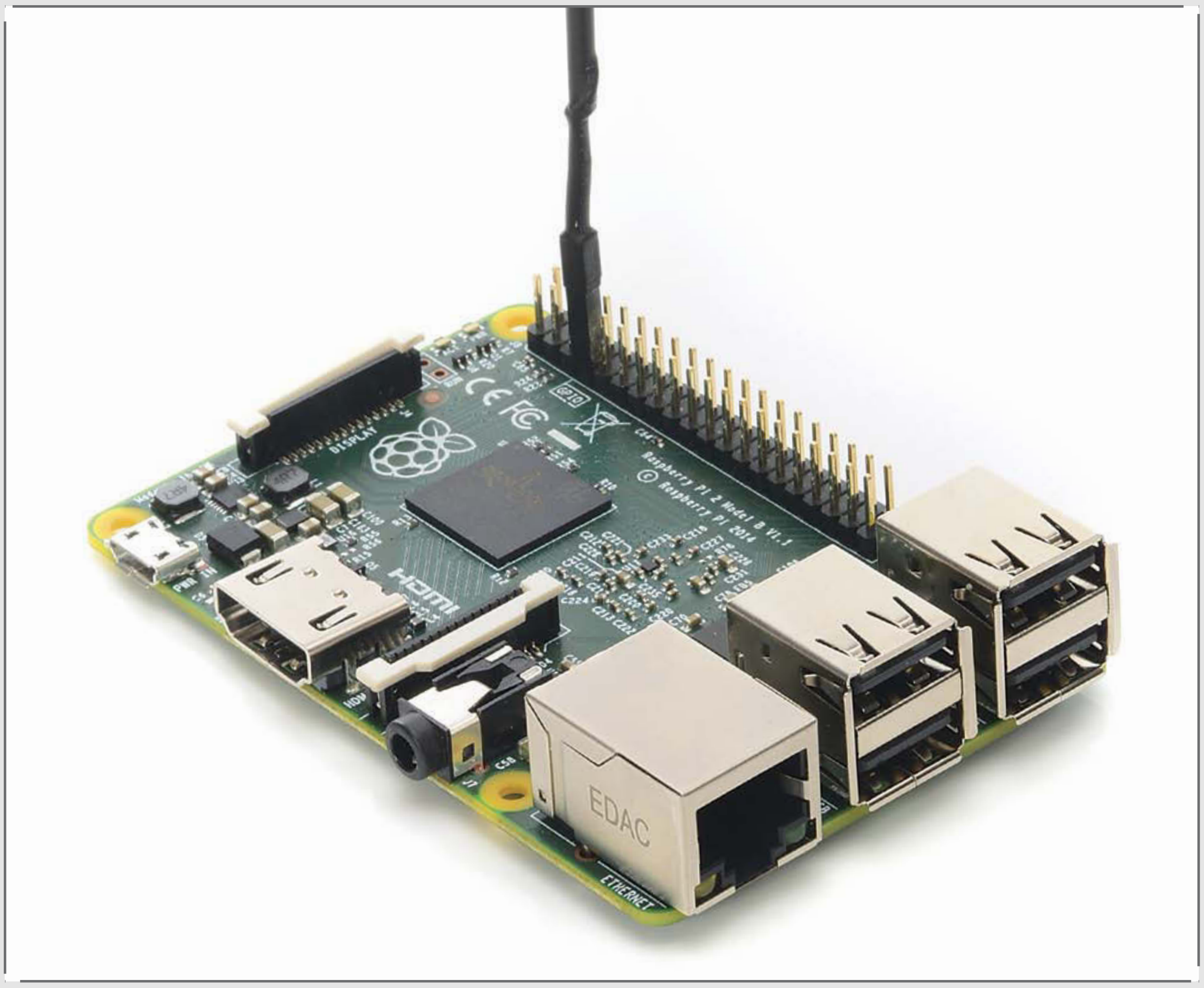
There's no facility for live broadcast. Instead, you will need to arrange your MP3 files in advance. These will be played using a script and transmitted to the nearest radio.

Planning on some DJ-ing? All is not lost, as you can produce your own MP3 files using Audacity or similar open source audio production tools. To get the files to play in a particular order, number them, or the folders, sequentially.

03 Power your Raspberry Pi radio

Planning to try this out in a remote area? You'll need a battery pack to power your Raspberry Pi, or else run it from your car's cigarette lighter. For flexibility, you may also need a mobile device with SSH software to wirelessly connect to your Raspberry Pi pirate radio in headless mode.

Of course, the project doesn't have to be portable and you can power your Raspberry Pi as usual.



04 Prepare your broadcast antenna

To broadcast from your Raspberry Pi, you'll need a suitable antenna. Electronic retailers stock copper wire that's around 2mm in diameter, but this is usually only available in bulk. As you only need a single 200-250mm length, the best option is to use a handheld rotary blade (or hacksaw) to cut the length from a wire coat hanger.

05 Connect the jumper

Strip the wire from a female jumper, leaving enough to solder the 2mm wire to.

Once cooled, add a 50mm length of heat shrink tubing to the top of the jumper and the lower portion of the antenna to insulate the connection. This will tighten as you apply heat from a hairdryer or heat gun. Be careful when heating, as the antenna will warm up and can burn your fingers.

Build a case

Okay, so you've already got a suitable case for your Raspberry Pi, but why not go all-out and put together a new case for this project? One idea is to take inspiration from the broadcast motif and design an old-style antenna case, with the Pi and the genuine antenna cleverly hidden inside it. Alternatively, a Mason jar (or other suitably wide-necked jar) will also make a great home for the PiFM – just drill a hole in the lid for the antenna!

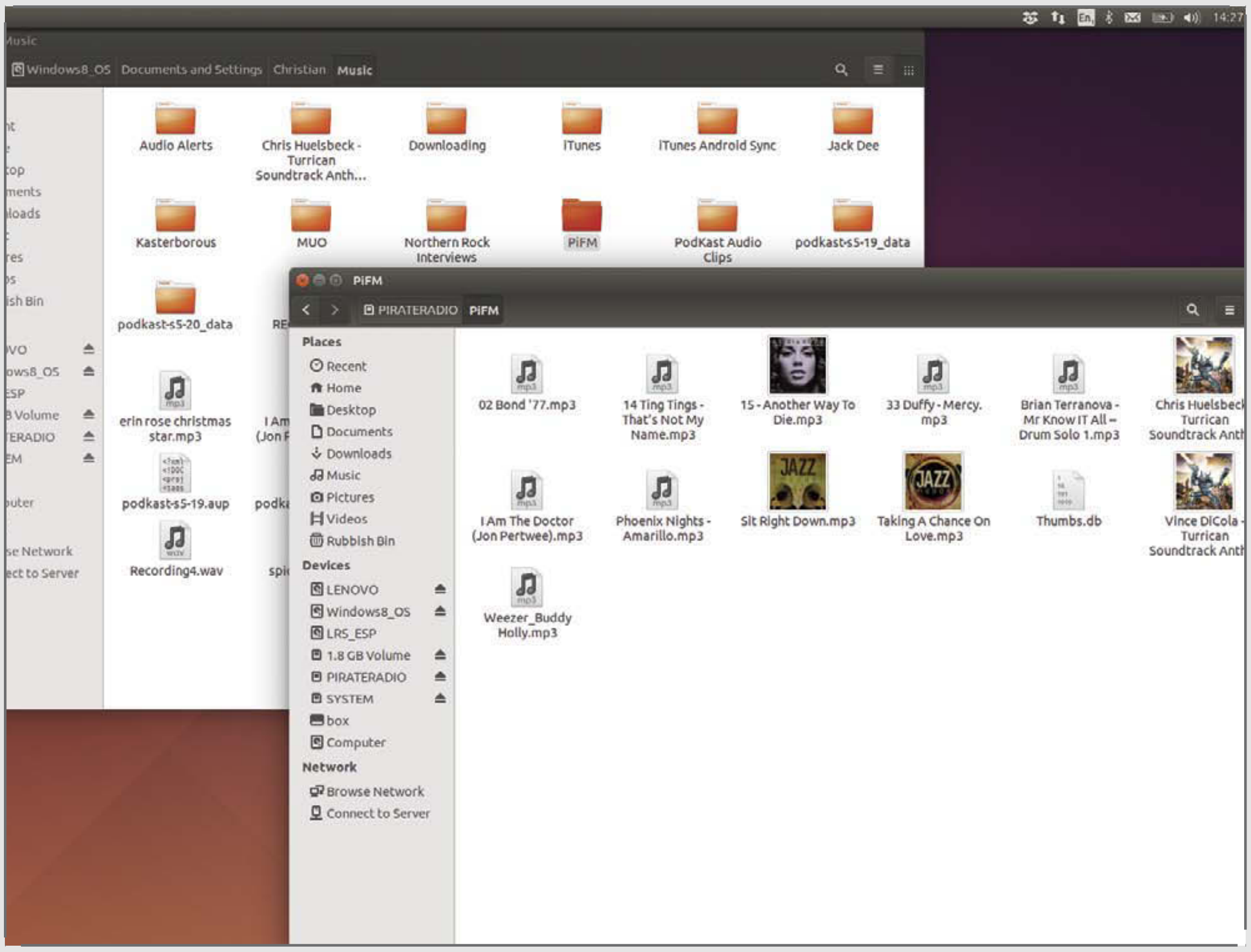
06 Connect the antenna

The antenna is connected to pin 4 on your Raspberry Pi's GPIO. Before you do this, check that your Raspberry Pi case is suitable for high profile GPIO connections. If not, consider connecting it to a short length of wire and mounting it on top of your case with an adhesive like Sugru, or perhaps just tape it to the side of your case. As long as the antenna has a connection to pin 4, you're good to go.

07 Prepare your SD card

As with all Raspberry Pi projects, it's good to start with a freshly flashed SD card. To get started quickly, use the disk image linked in the kit list, extract the ISO file and flash. However, if you would rather spend some time tweaking the script, flash Raspbian Wheezy and install PirateRadio.py from GitHub.

"Unlicensed broadcasting on the FM band is an offence. This tutorial is merely a proof of concept – one that might be used for a school radio project, for instance"



08 Copy your MP3 files to SD

You cannot simply dump your MP3 files on the SD card. With your flashed SD card still inserted into your PC card reader, browse to the /Pirate Radio partition of the card and paste your copied files.

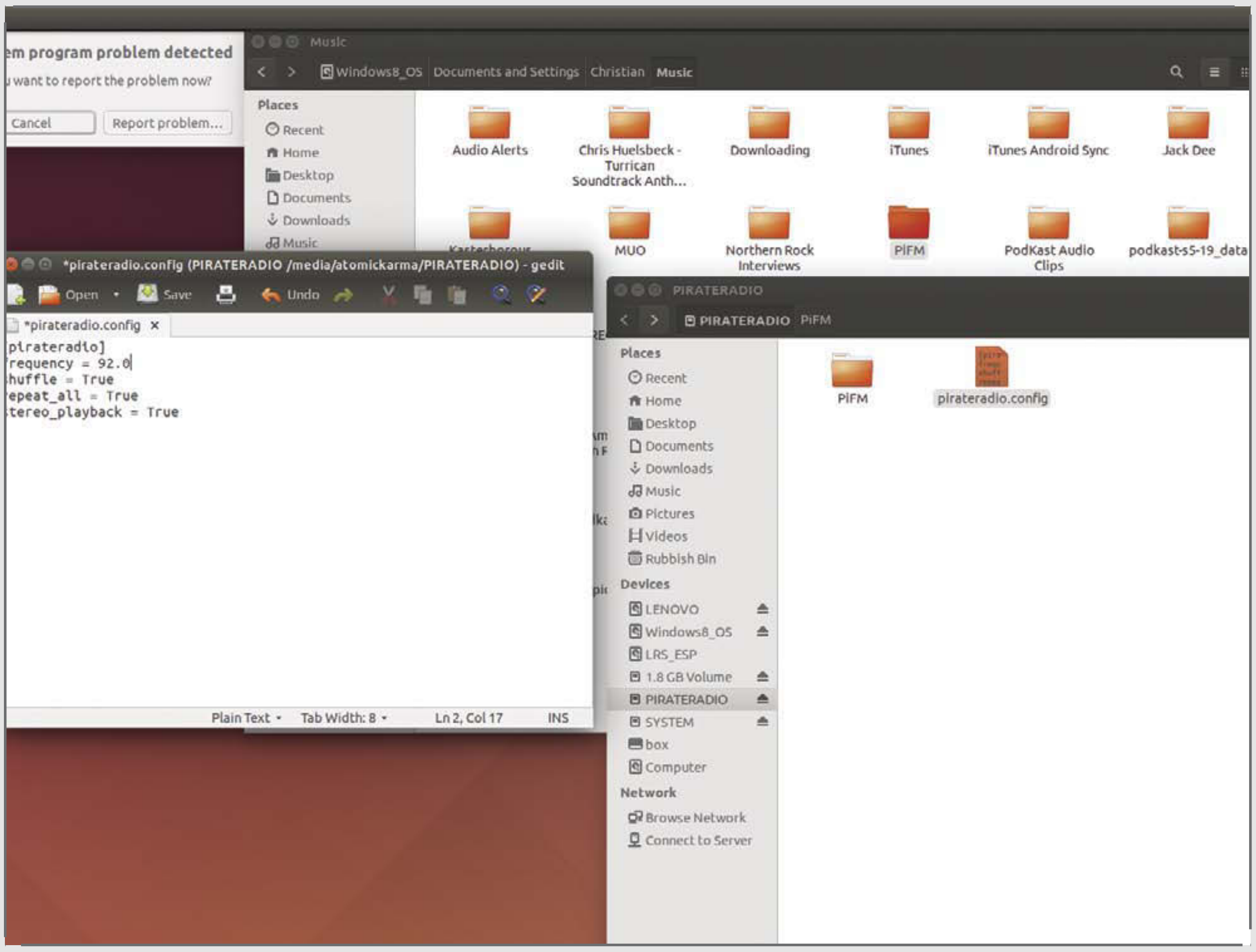
Beyond simply pasting in numbered MP3s, you can also drop in named folders from your music collection containing entire albums or artist catalogues.

09 Configure the Pirate Radio

With the SD card still inserted into your PC, open the pirateradio.config file in a text editor. Look for the frequency setting and adjust this as necessary. You'll need to set this to a frequency that is currently unoccupied, so switch on your FM radio, find some free space and change the pirateradio.config file as necessary. Save and exit the file when you're done.

Multiple audio formats

Throughout the tutorial, we've talked about audio files as MP3s, but one of the many beauties of the PiFM project is that it supports other formats. These are re-encoded as required in time for broadcast, based on a playlist created when the Python code scans the SD card for audio files. In addition to MP3s, you can cue up and broadcast files in FLAC, WAV, M4A, AAC and WMA.



10 Random and continuous music

Should you plan to add a lot of music to your SD card for playback on your pirate radio project, you may want to use the shuffle and repeat_all settings in the pirateradio.config file. By default these are set to true, but to disable, you simply need to change true to false. Save when you're done, and remember to unmount the SD card before removing it from your computer.

11 Stereo or mono?

The pirateradio.config file offers you the choice of setting a true or false value to the stereo_playback value. You should consider this carefully, as it will determine quality and range for your broadcast. Set to true, the broadcast will be of superior audio quality. However, the range will be reduced as additional power is required. So setting stereo_playback to false will increase your broadcast's range.

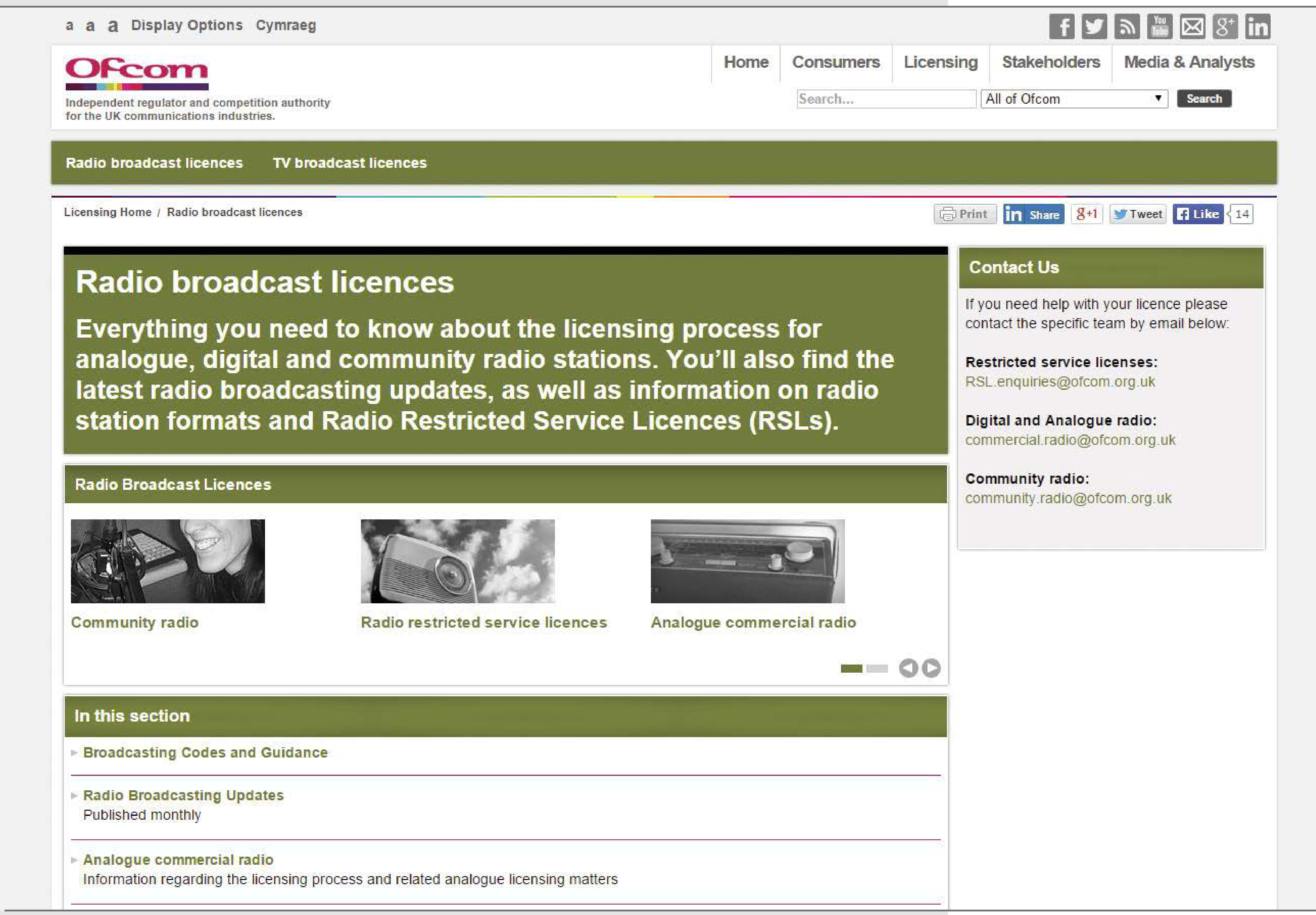
> "This project is perfect for short range use, such as playing your MP3s on an old car radio, although you'll have to modify the length of the antenna for this"

However, due to the scale of the transmitter, this is only a matter of 20 feet!

12 Appreciate the law

It's all too easy to get into trouble with this project, so before you start broadcasting make sure you're familiar with the law and licensing requirements for broadcasting on the FM band. This project is perfect for short range use, such as playing your MP3s on an old car radio, although you'll have to modify the length of the antenna for this. Schools may also benefit from a Raspberry Pi radio station.

Below Check out local radio broadcasting licenses for more details on the law regarding broadcasting music



13 Take to the airwaves!

Insert your SD card into the Raspberry Pi and plug the device in. Meanwhile, step away from the device and head into the next room with your FM radio and tune into the specified frequency. Once the Raspberry Pi has booted you should find that the MP3 files are being broadcast!

14 Troubleshooting bad broadcasts

Problems with your broadcast? Check the FM radio is capable of picking up other stations. If you're in the UK, look for Radio 2 on 88-91 FM, which can be received virtually anywhere.

You should also check the frequency setting in the pirateradio.config file. Remember that the band is accessible in the UK from 87.5-108 FM. As such, you cannot access frequencies beyond these points on an FM radio.

15 Check your antenna for problems

Reception issues may be traced back to the antenna. Double-check the soldering is secure and confirm that the wire is connected to GPIO 4. Using the wrong pin won't harm your Raspberry Pi, but equally it won't result in audio being broadcast to your FM radio!

Interference in the broadcast may be due to wireless routers and microwave ovens. Your Raspberry Pi's power source may also cause problems.

16 Elevate your broadcast

You can improve the range by positioning your Raspberry Pi and the PiFM antenna in an elevated

How does the Pi broadcast?

Spread-spectrum clock signals on the GPIO pins are the secret power behind the Raspberry Pi's surprising hidden ability to broadcast on the FM band. By utilising this energy with an antenna on pin 4, you can turn a method employed to reduce electrical interference with other devices connected to and situated near your Raspberry Pi into a tool for radio communication.

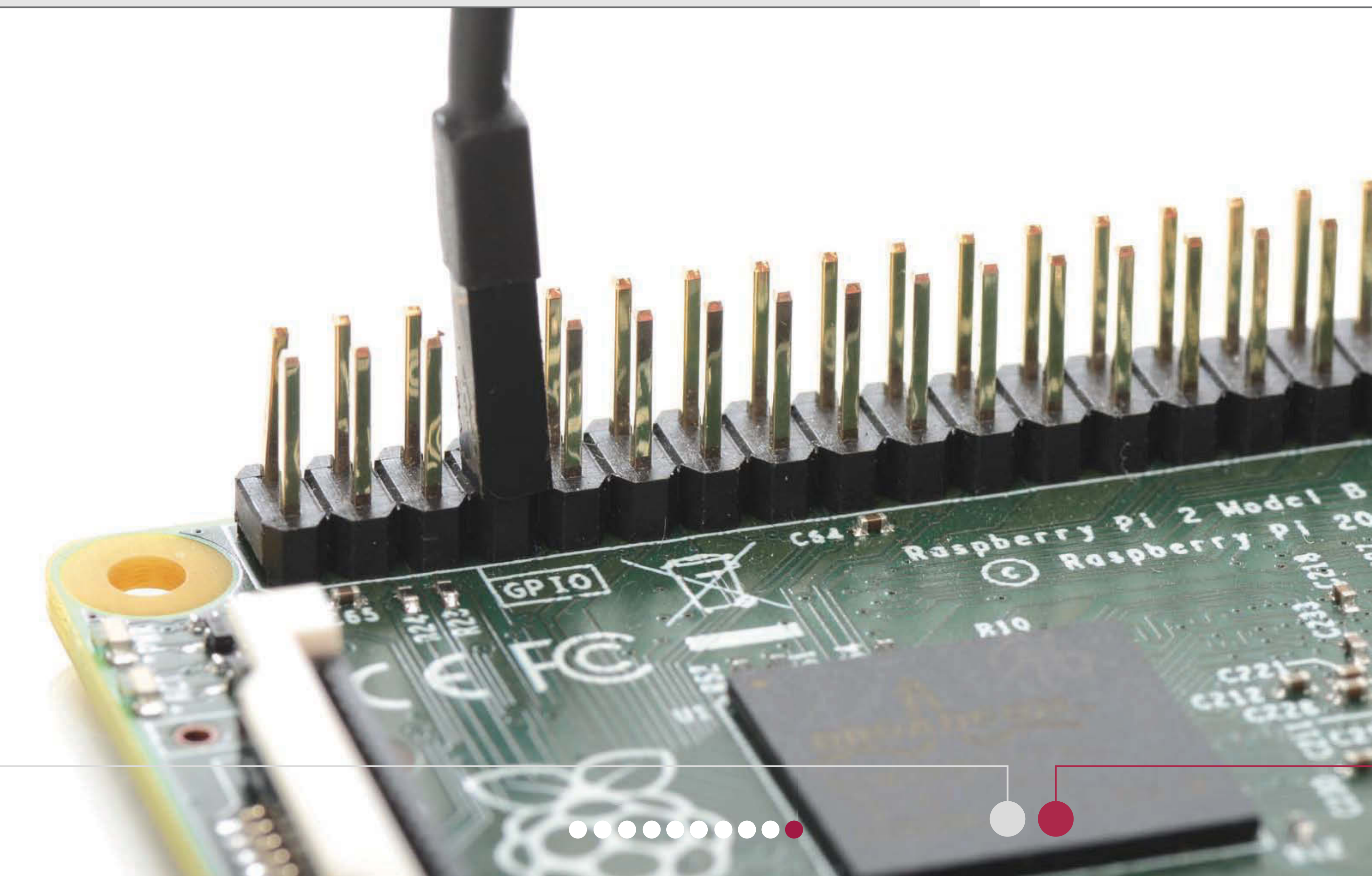
position. You might, for instance, place it by an upstairs window.

Even better results can be achieved by broadcasting from up a tree or on a hillside – but keep in mind that an unlicensed project really needs to maintain a short range.

17 Curb your piracy

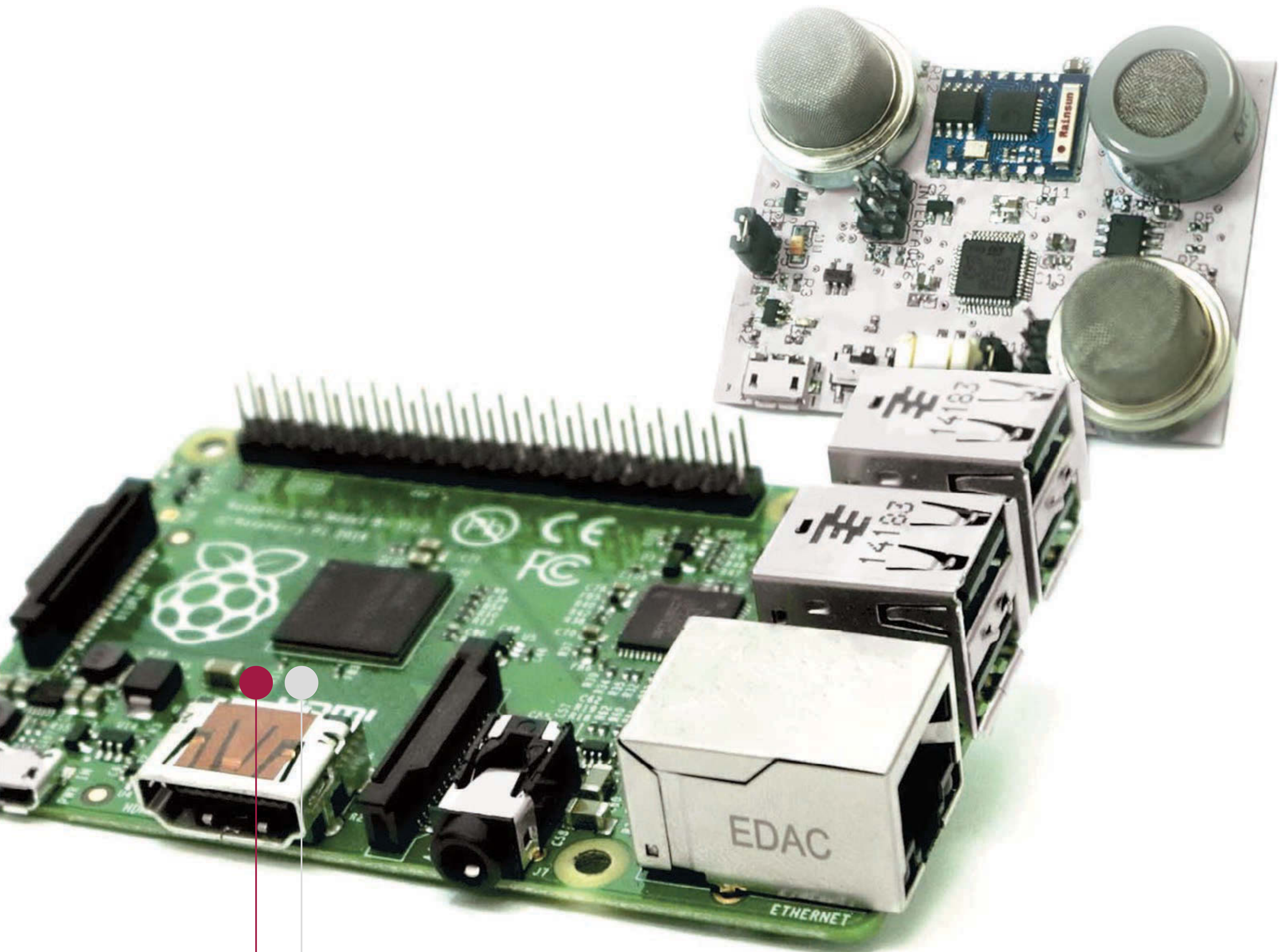
You've built a compact, potentially portable radio transmitter, but remember that it isn't the 1960s and you're not a revolutionary. Consider the approved uses for the project and stick to these, rather than using it to cause mischief and get a criminal record. You could make an FM repeater, for example, or create a wireless mic. If you really want to get on the radio, consider volunteering with your local community station.

Below The original GPIO 4 hack was discovered by two students at Imperial College London



Environmental science with the Sensly HAT

Conduct experiments, monitor pollutants and more with this clever little module for your Raspberry Pi



The Sensly is a smart module attached to the Raspberry Pi. It has its own microprocessor that can handle analog data and handle sampling from various sensors without the need to waste your Raspberry Pi's processing power. There are three different gas sensors attached to the board, allowing you to sense a multitude of gases, as well as humidity and temperature sensors. Plus, this Raspberry Pi HAT can be easily extended with its array of analog ports and an I2C interface.

In keeping with the Raspberry Pi way of doing things, the module also provides a Python API that does all of the hard work for you. All of the maths, calibration and error correction is handled by the API itself, allowing you to play around with meaningful data immediately.

THE PROJECT ESSENTIALS

Sensly HAT
http://sensly.uk

01 Setting up your Sensly HAT

This tutorial assumes you have set up your Raspberry Pi and it is connected to the internet. So, the first step is to plug your Sensly HAT into the Raspberry Pi header. Once you have done this, power on the Raspberry Pi and type the following commands into the terminal:

```
# Install git so we can download the API
sudo apt-get install git
git clone https://github.com/Altitude-Tech/
   SenslyPi.git
cd SenslyPi
# Install the python API
sudo python setup.py install
```

02 Testing the Sensly

The first step is ensure the Raspberry Pi can communicate with the Sensly HAT and make sure it

is functioning correctly. Type the following command and it will respond with "ok", and give a report if everything is working:

```
# Run test sequence on the Sensly
sensly-test
```

03 Using the API

We are now going to use a terminal-based text editor to write some code – we're going to use Nano but you can use any editor you like. The following command will create a file called sensly.py and open it for editing.

```
nano sensly.py
```

You can type the following program to get basic data from the Sensly.

```
import os
from sensly import Gases

atmosphere = Gases()

while True:
    print(“Humidity Level:”)
    print( str(atmosphere.humidity()) )
    print(“Temperature:”)
    print( str(atmosphere.temp()) )
    time.sleep(5)
```

04 Getting started with gas sensors

To start using the sensors, it gets a little bit more complicated as we need to preheat the sensors to ensure the readings from the Sensly are stable. To

do this we use a loop in the code which waits until the preheating sequence has finished. This takes two to three minutes if you have only just powered up the Sensly.

```
import os
from sensly import Sensly

sensly = Sensly()

# Wait until preheating has finished
# so we can get stable readings
While (sensly.preheated() == False):
    print("Preheating . . . . .")
    time.sleep(1)

print("Sensly is ready to read pollution
    levels!")
```

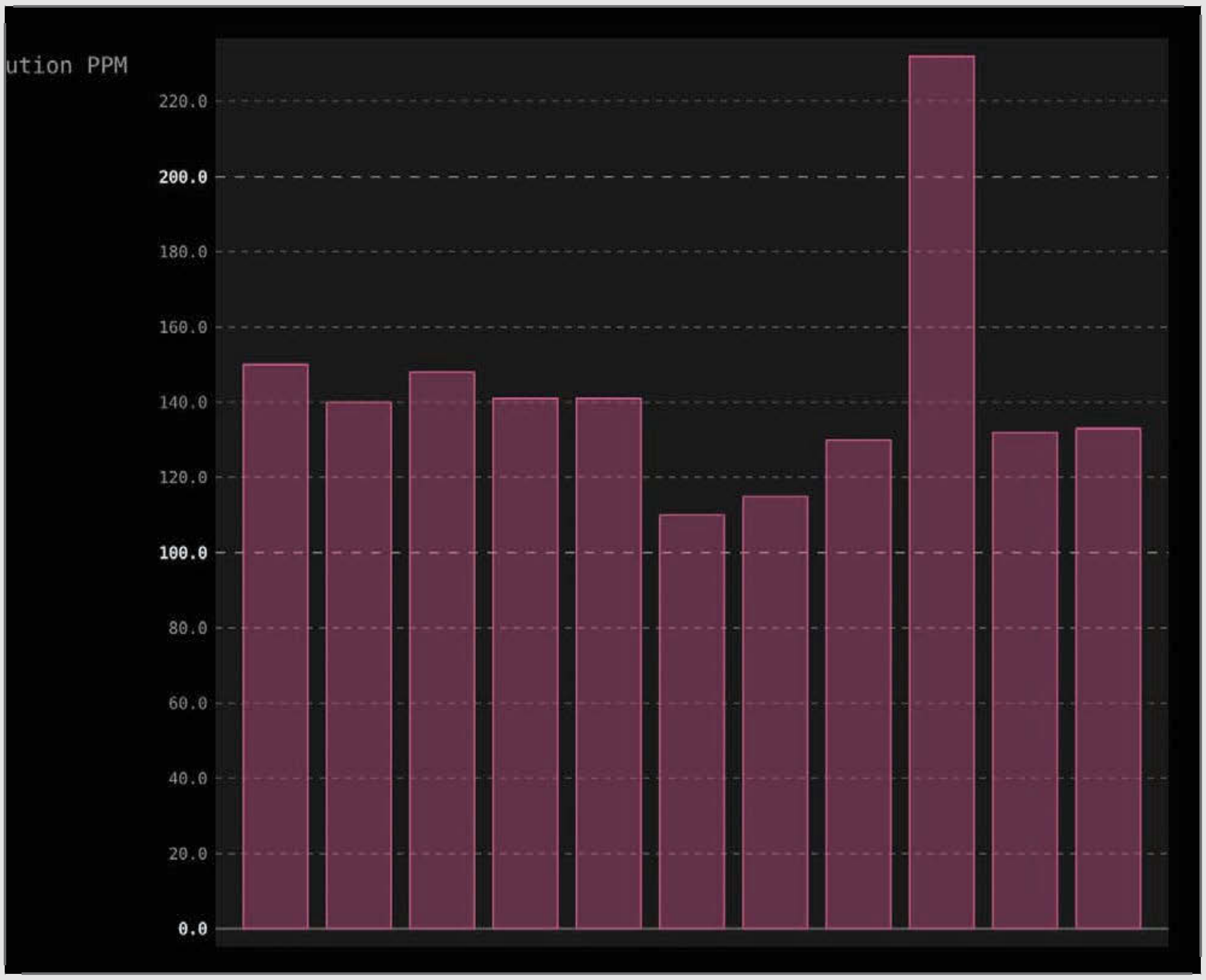
05 Getting pollution data

Now that we know how to get the gas sensors ready to read data, it's time to start taking some measurements. Add the following code to the previous program:

```
While(True):
    # Print the general level of pollution
    print("General pollution level (PPM)")
    print( str( atmosphere.pollution()) )

    # Print the level of industrial pollution
    # (e.g. benzene, nitrogen oxides)
    print("Industrial Pollution (PPM)")
    print( str( atmosphere.pollution.
    industrial()) )
```

Unlocking the ARM cortex

The brain of this add-on board is its ARM cortex, which handles all of the analogue signals, data sampling and communications to the Pi. This is a powerful little chip with many features and its firmware is open source. As a result, you can flash reprogram the HAT from your Raspberry Pi, giving you the flexibility of a microcontroller like the Arduino but with all the advantages of a full operating system – making this a very fun little HAT to tinker about with.



Left Pygal is a dynamic SVG charting library that offers Python/CSS styling by means of preset themes

```
# Print the Humidity
print("Humidity")
print( str( atmosphere.humidity()) )

time.sleep(20)
```

06 Adding custom sensors

The Sensly provides five ports for additional sensors, which can easily be controlled with a simple Python script. They are particularly useful because the Raspberry Pi does not provide any native analog ports. This allows you to add extra sensors to suit your projects, such as an LDR to measure light pollution. Using the following script you can easily read analog and digital data from the expansion ports:

```
import sensly
from sensly import Port
```

```
sensor1 = Port(Sensly.port1)

print( str( sensor1.analog_read()) )
```

07 Visualising your data

For this we are going to use a great open source project called pygal, which makes it easy to generate cool vector graphs from your data. First of all we need to install pygal, and then we can write some code to generate some information on pollution levels. Run the following commands to set up pygal:

```
sudo apt-get install python-setuptools
sudo easy_install pygal
```

The following code will sample the air for ten hours and generate a vector graph, although you can edit the timings to suit your needs:

```
import pygal
import os
from sensly import Gases

atmosphere = Gases()
graph = pygal.Bar()

samples = []
for i in range(10):
  samples.append(atmosphere.pollution.
    industrial())

graph.add('pollution', samples)
graph.render_to_file('pollution.svg') #
    Generate graph
```

Optimise Python

This article on Python optimisation looks at how to use external compiled code to speed your program up

In issue 21 of RasPi we looked at profiling code, to see where to apply your skills, and then optimise the existing Python code. In those cases, you have the option of outsourcing troublesome parts to an external language and compiling that code to machine language. The most common technique to do this is to use Cython (http://www.cython.org) to take C code, compile it to machine code, and then use it within your Python code. The first item you need is a C compiler that Cython can use. On Linux systems, the default compiler is GCC. To be sure that you have all of the tools, you can use the command:

```
sudo apt-get install build-essential
```

The second item you need is the Cython package for Python. If you want the latest version available in Pypi, you can use:

```
sudo pip install cython
```

… to install the Cython package into the system Python library location.

Before we dive into how to use Cython, we will confirm everything is working correctly. Your Cython code needs to be in its own file, with a .pyx file name ending. A Hello World function (in the file hello.pyx) would look like the following:

```
def hello_world(name):
    print(“Hello World to %s” % name)
```

You then need a setup.py file to define the compilation step:

```
from distutils.core import setup
from Cython.Build import cythonize
setup(name=‘Hello world app’, ext_
    modules=cythonize(“hello.pyx”),)
```

With these two files, you can build the external code with the command “python setup.py build_ext --inplace”. This creates a C source code file and compiles it into a shared binary file with a .so file name ending. You can then use this new external code with the Python command:

```
from hello import hello_world
```

… and use it as any other Python function.

While the above will let you take your Python code and move it out to compiled machine code, the real power comes when you use Cython to use C libraries within your Python code. The functions within the standard C library are already defined within Cython.

Why Python?

It’s the official language of the Raspberry Pi. Read the docs at https://www.python.org/doc/

You can import these functions with the cimport statement. If you wanted to use the function atoi() to convert a string to an integer, you could use:

```
from libc.stdlib cimport atoi
cdef parse_char_to_int(char* s):
    assert s is not NULL, "String is NULL"
    return atoi(s)
```

Cython also includes declarations for the C math library. You can import these from the libc.math package. While these two libraries are handy, the majority of the code you need will reside in other libraries. In these cases, you have to provide declarations of them yourself. As an example, we can look at using the sine function from the math library and do the importation manually. The first step is to provide a declaration:

```
cdef extern from math.h:
    double sin(double x)
```

This declaration code resides in a file with the ending .pxd. This file needs to have a different name than any .pyx files. As an example, you might place the above code in a file named csine.pxd. You can the import it within a file named sine.pyx:

```
cimport csine
```

… and use it within your Cython code. If you use an object, you can use the function __cinit__ to handle memory management issues. When you are done using an instance of this object, use the __dealloc__ function to clean up steps and memory frees. Also tell Cython which external libraries need to be linked in. You can do this with the extra option:

```
libraries=["libname"]
```

… added to the Extension entry in the setup.py file.

Within Python, you can create new objects without worrying about where they will be stored, and you can discard with equal impunity. But sometimes, when you are including C code, you need control over memory management. Cython includes declarations for the functions malloc, realloc and free from the C standard library. For example, let's say you want space for an array of doubles. You can do this with:

```
cdef double *my_array = <double
    *>malloc(number * sizeof(double))
```

When you are done with this array, you can clean up with:

```
free(my_array)
```

The problem is that this memory is outside of the regular Python heap and so is unaccounted for. A preferred method is to use the C-API functions provided in the package cpython.mem. The equivalent functions are PyMem_Malloc, PyMem_Realloc and PyMem_Free. These have the same usage and interface as the lower level C functions from the standard C library. Of course, once you start down this road, you are responsible for freeing memory and avoiding memory leaks.

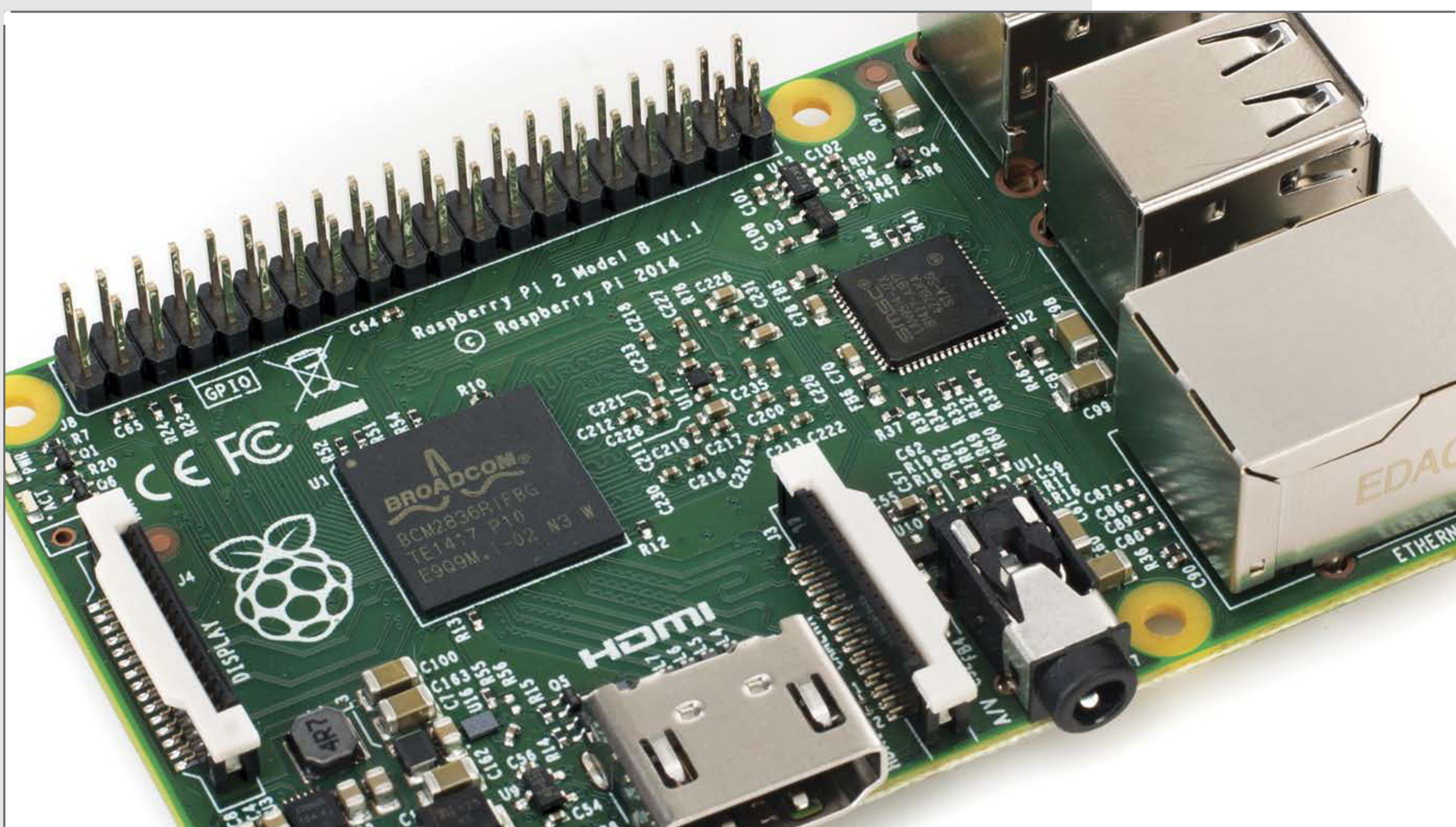
Now you've seen how to start adding C code to your Python code, you can start some major optimisation. This is actually how packages like numpy and scipy get their impressive speeds. And now you can apply these same techniques to your own code. However, as always, you need to balance the amount of work it takes to write the code with the amount of work it takes to maintain the code and the amount of speedup you get. Try to avoid the temptation of over-optimisation.

```
# The contents here should be divided into several
# separate files in order to use the examples

# hello.pyx
# Basic Cython hello world
def hello_world(name):
    print("Hello World to %s" % name)

# setup1.py
# This is the setup Python script to compile the
# Hello World Cython example
from distutils.core import setup
from Cython.Build import cythonize
    setup(name='Hello world app', ext_
modules=cythonize("hello.pyx"),)

# c_atoi.pyx
# You can import standard Library functions directly
from libc.stdlib cimport atoi
cdef parse_char_to_int(char* s):
    assert s is not NULL, "String is NULL"
    return atoi(s)

# csine.pxd
# You need to declare external C library functions
# to use them within Python
cdef extern from "math.h":
    double sin(double x)
```

```
# sine.pyx
# You can then import and use the C function
cimport csine
csine.sin(45.6)

# setup3.py
# You need to add in which external libraries to link to
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize
ext_modules=[Extension(“sine”, sources=[“sine.pyx”],
libraries=[“m”])]
setup(name = “Sine”, ext_modules = cythonize(ext_
modules))
```

Cython and Pyximport

One major issue with using Cython is that you are dragged back into the development cycle of compiled languages. In order to test new code, you need to write, compile, then run these changes in order to see how they behave. If the code you are writing isn't too complicated, then a different option you can choose is to use pyximport. Pyximport is provided through the Cython package, and enables you to have your code compiled on the fly when you need to use it. In order to use it, you need to import it and run the install function:

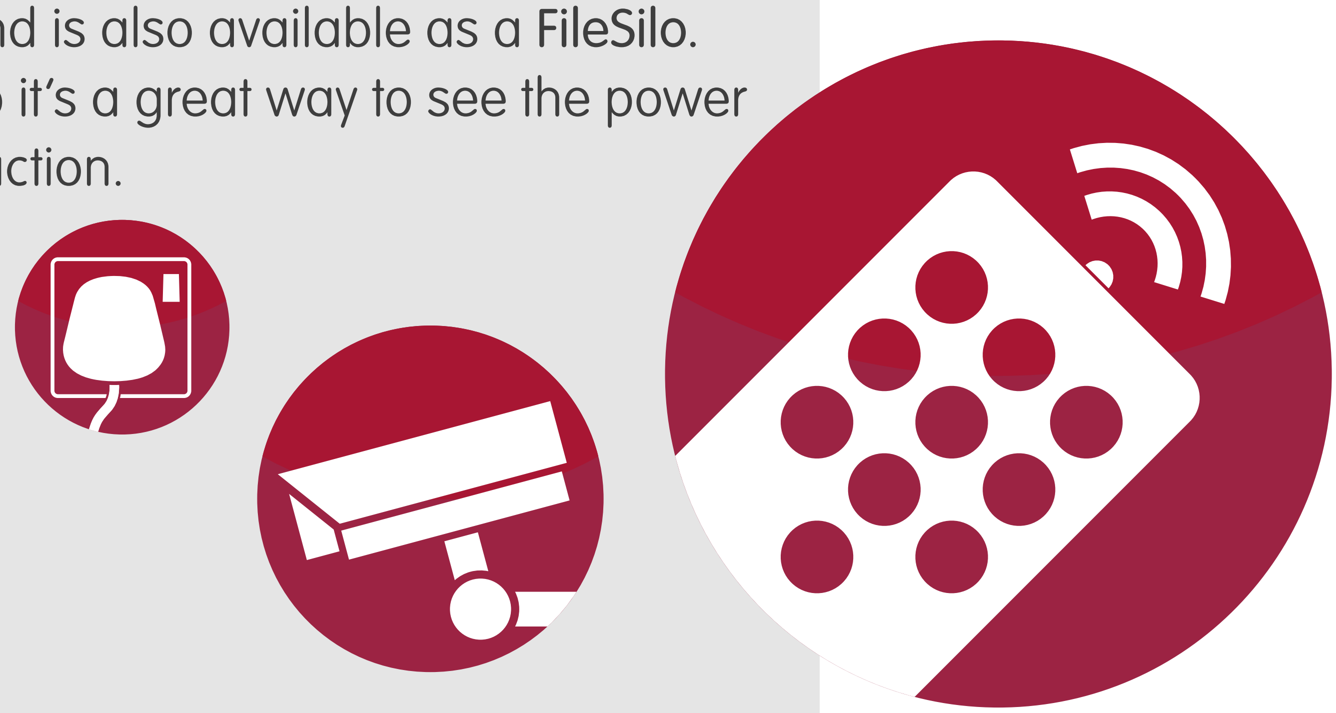


```
import pyximport; pyximport.install().
```

Then, when you import your pyx file, it gets silently compiled. However, one issue is that you don't have much control over how this compilation is handled. In most cases, though, the defaults are usually fine. Pyximport behaves like make, in that it only does a recompile when a source file is newer than the associated source file. In simple cases, you usually just have a single PYX file. If you have multiple dependencies, you can delineate them within a file with the ending .pyxdep. Each dependency should be on a separate line within this file.

Coding with Cython

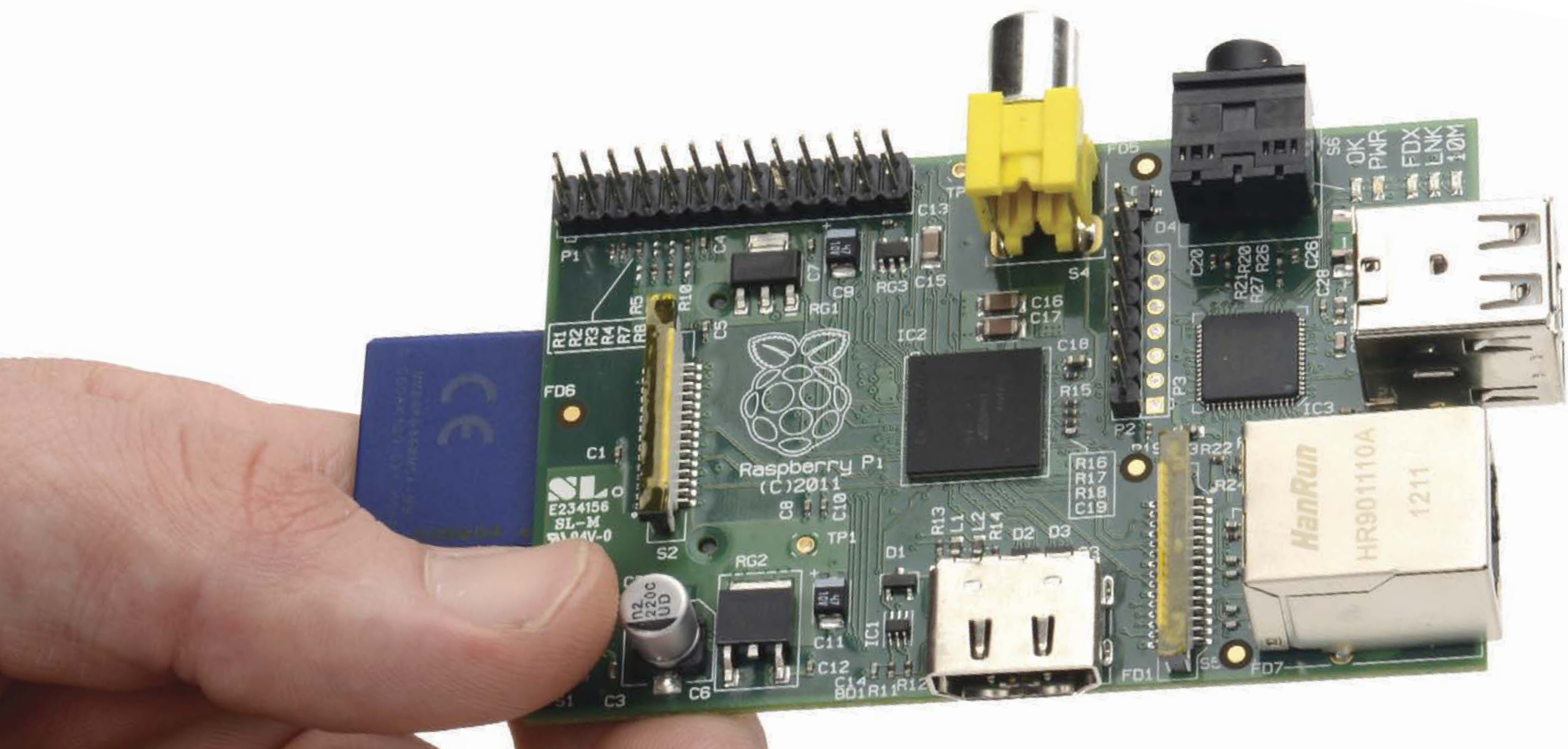
If you're interested in learning more about Cython, it's worth picking up a copy of issue 153 of Linux User and Developer – you can order one from http://bit.ly/1M2fWKG. In this issue, Liam Fraser explains how to write a simple polyphonic synthesiser using Python and Cython in tandem, taking advantage of Cython's improved performance in order to effectively play multiple notes at the same time. The code is listed in its entirety for you to examine and type up yourself, and is also available as a FileSilo.co.uk download, so it's a great way to see the power of this compiler in action.



Talking Pi

Join the conversation at…

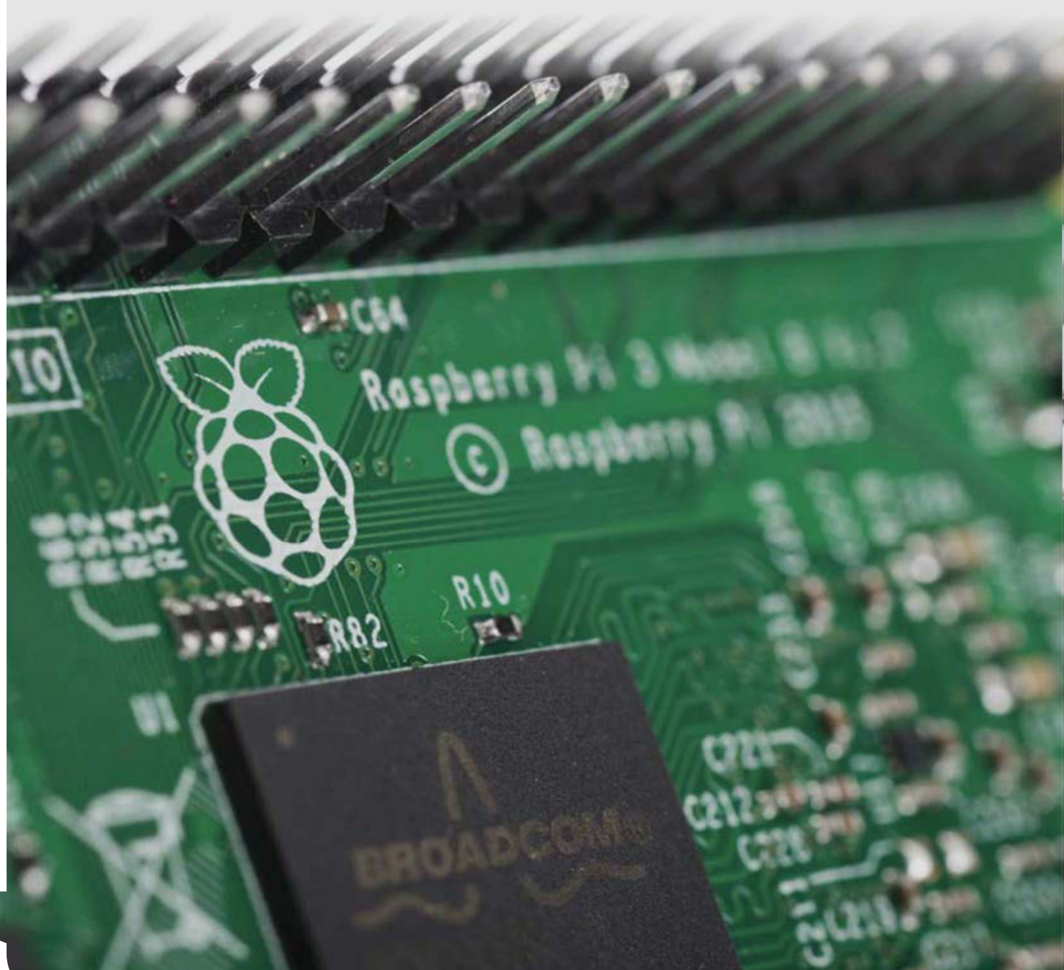
@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easily as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

When I try to run my new Pi, the LEDs flash like they're supposed to but nothing appears on the screen at all.
Joe via email

Sometimes the SD card you're using can be faulty and cause issues like this, although when the SD card is to blame then the LEDs won't usually light up, so we reckon that the problem might be down to the cable you're using to connect the Pi to your display. First do the obvious checks like making sure that there are no loose connections in either end of the cable and checking that the integrity of the cable itself is still solid. If you're still having problems, connect a keyboard to your Pi and screen setup and hold down the 2 key if you're using a HDMI cable (Hold 3 for composite and 4 for NTSC) as this will force the Pi to look for the connection at startup.



Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

JUST A SCORE

WHAT'S YOUR JUST A SCORE?

Have you heard of Just A Score? It's a new, completely free app that gives you all the latest review scores. You can score anything in the world, like and share scores, follow scorers for your favourite topics and much more. And it's really good fun!

Can I put my old SD card and the OS and data on it into my new Pi?
Tom via email

Yes you can, but you'll probably need to update the OS for your newer Pi, especially if you're running Raspbian. In Raspbian, simply type:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
```

Or head over to **https://www.raspberrypi.org/downloads/** to grab the latest version of the OS.

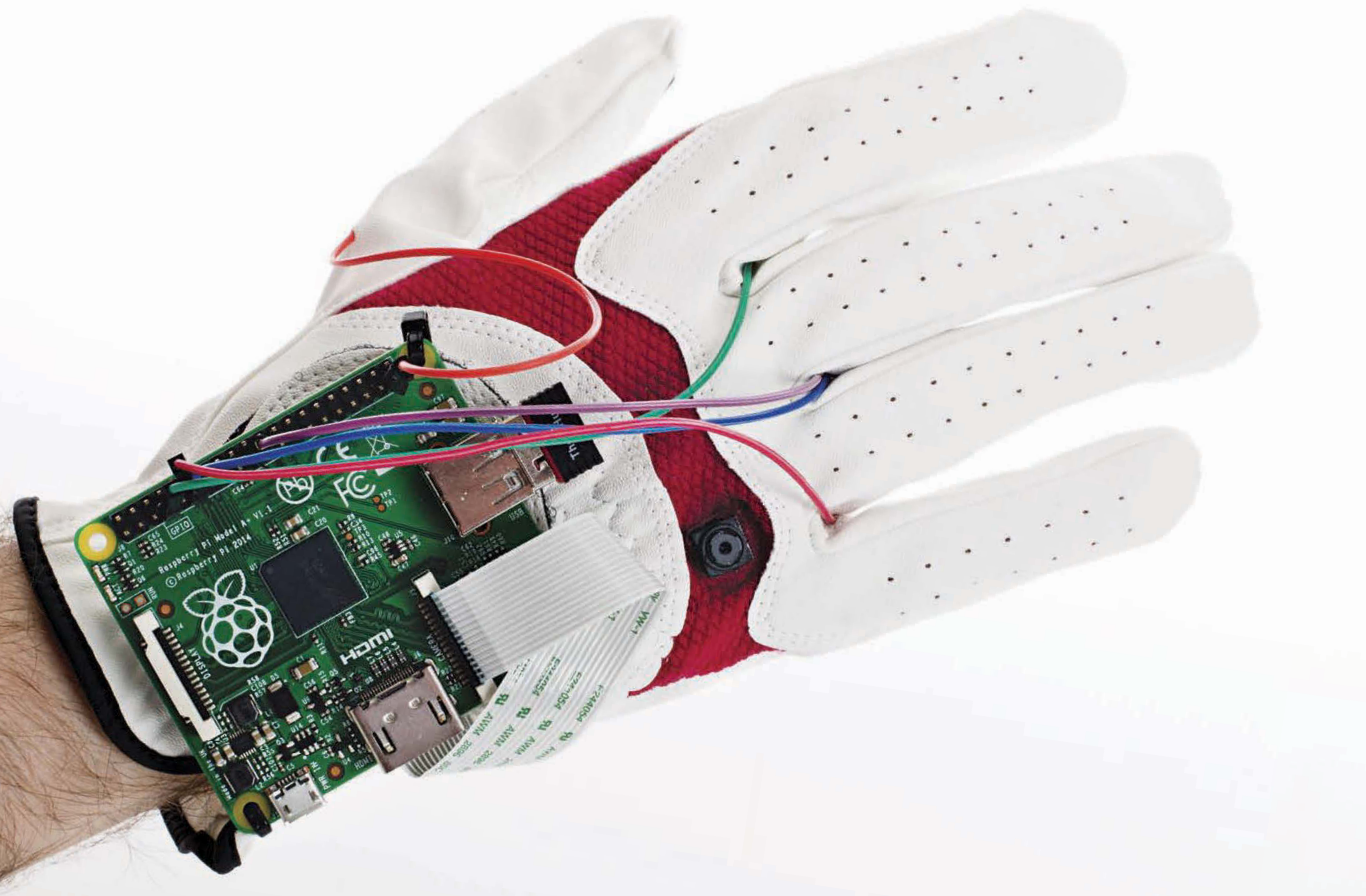
What's the small red square that keeps appearing in the top-right?
Chris via email

That's a bad sign Chris – it means that your PI is too hot! There are several reasons why your Pi might be overheating. If the red square only appears sporadically it indicates that something is making the Pi heat up every so often. How much are you asking it to do? If it's performing a task with a heavy load or you're pushing it to do something quickly, then that may well be the source of the overheating. If the red square is there all the time however, then it may indicate that you're putting too much voltage through your Pi via the power supply or that there's a fault with one or more of the components.

Next issue

Get inspired · Expert advice · Easy-to-follow guides

Build a Pi glove



Get this issue's source code at:

www.linuxuser.co.uk/raspicode